EVENT BASED TRANSIENT NOTIFICATION ARCHITECTURE AND NoSQL SOLUTION FOR ASTRONOMICAL DATA MANAGEMENT

A thesis presented in partial
fulfilment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE

at Massey University,
Albany (Auckland), New Zealand.

MASSEY UNIVERSITY
NEW ZEALAND

Yu Zhao

2015

Abstract

This thesis describes the development of a messaging system for astrophysical transient event notifications, and explores the use of NoSQL databases for astronomical data management. The Microlensing Observations in Astrophysics (MOA) Project is used as a case study for which respective implementations are provided.

The Java Message Service (JMS) was chosen as the most suitable message implementation. Full implementation details of this new generation transient event notification architecture are described. It was found that the JMS-based architecture can provide a high performance on data delivery and could be up-scaled to meet high event rate situations.

For NoSQL databasing, it was found that using a document-oriented database for astronomical data management is much easier than traditional relational databases, and significantly improves overall data access performance. In addition, a NoSQL database can be very easy to scale up in order to meet demands of large workload, and the scaling approach is very cost effective. Full design details of using a document-oriented NoSQL database for astronomical data management are presented in this thesis.

The approaches developed in this research will provide scalable and efficient solutions for resolving similar computational challenges in astronomy. There are great opportunities to further extend this research and to make additional contributions to the field.

Acknowledgements

I would like to thank my supervisor Ian Bond and my co-supervisor Winston Sweatman for their guidance over these four years. This thesis could not have been completed without their support, encouragement and help. I enjoyed every meeting and discussions of my research with them; they give me lots of fun during my PhD life.

I would also like to thank everyone at the Institute of Natural and Mathematical Sciences for their inspirational seminars, discussions, suggestions and helps.

My family is very important to me and I could not complete this thesis without their continued support. Special thanks to my mother for her sacrifices over the years in New Zealand. I would also like to especially thank my father for his support.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Time Critical Astrophysics

There are many astrophysics phenomena occurring repeatedly in the universe. Most of these phenomena are predictable and they can be observed at any time. There are, however, some phenomena which are short-lived and only possible to observe on particular timescales, such as nova, supernova, variable stars, comets and gamma ray bursts, among others; these types of phenomena will occur over very short periods (normally only over a few hours, days, or weeks) and quickly disappear. Therefore, these types of phenomena are also referred to as “transient events” [1] [2]. Generally, an astronomical transient event has three common characteristics, as outlined below:

- Transient; each transient event only lasts from a few hours up to few days [3], although some transient events may last longer (e.g., a few months), but these are very unusual cases;

- Unpredictable; it is not possible to predict when and where a transient event will occur in the universe. The only way to detect transient events is through using a telescope to observe millions of stars in the sky night by night, in order to find potential transient events; and

- Unrepeatable; each transient event is a one-time-only event. In other words, a particular transient event will not occur again after it has disappeared. This means that if a transient event is not observed on its first occurrence, then astronomers and observers have no second chance to observe the same transient event.

As there is a time limitation on observing a transient event, transient astronomical events are also referred to as time critical (or time domain) astrophysics. When a new transient event has been discovered, the new transient event data needs to be alerted to the global astronomical transient research communities for detailed follow-up observation and further study.

1.2 Gravitational Microlensing

Gravitational microlensing [4] is one well known transient astronomical phenomenon. It is very useful for detecting planets and other objects in an extra-solar system which emit only a little light, or even no light; these objects range from masses of planets to masses of stars [5]. Without gravitational microlensing astronomers are only able to detect those bright objects (planets or stars) that emit a great amount of light, or large objects that are able to block background light (clouds of gas, dust, etc.). Gravitational microlensing is a very helpful technique for astronomers studying and understanding distant planets and faint objects in the extra-solar system.

The first person to introduce the concept of gravitational microlensing was Albert Einstein in 1936. He wrote a paper [6] that described how the gravitational microlensing effect, and also stated that there was “no chance of ever observing this phenomenon” [6] [8].

Very specific conditions are required in order to trigger a gravitational microlensing event. It requires a background source object, a foreground lens object (assuming that both objects are the mass of a star), and the observers on Earth to all be nearly perfectly, or perfectly, aligned. The occurrence of a gravitational microlensing event is very rare [194] [196] [197]. The Galactic Bulge (GB) was proposed as a target field for microlensing searches, because the Galactic Bulge (GB) region contains very high star densities (more than a hundred-million stars [195]), and high star densities may increase the rate at which gravitational microlensing events are triggered. However, even though the Galactic Bulge (GB) region contains very high star densities, the occurrence probability of a microlensing event is still extremely rare ($\approx 10^{-6}$[194]), and observers need to observe the bulge region and monitor changes in the brightness of stars for several months to obtain significant search results.

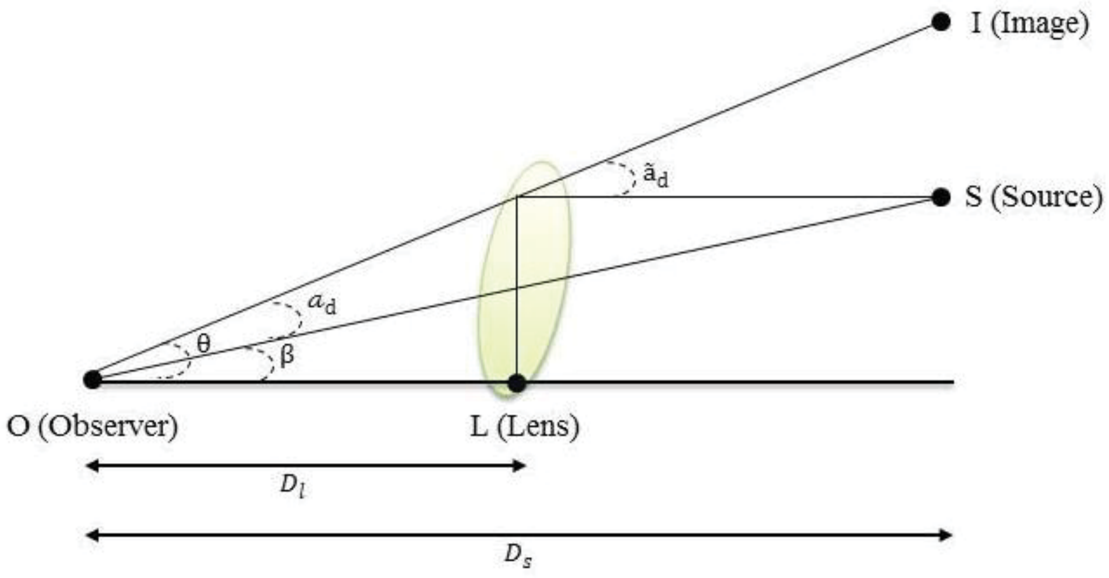


Figure 1: Microlensing geometry. D_l is used to separate the observer O and Lens L, and D_s is the length of the distance from the observer to the source S. The green shaded area in the middle represents the gravitational field. The light emitted from the source star will be bent by this gravitational field.

Figure 1 shows the basic geometry of the gravitational microlens and the equation for the deflection angle is:

$$\tilde{a}_d = \frac{4GM}{rEc^2}$$

where $\tilde{a}_d$ is the deflection angle and rE is the Einstein radius. The source location will be displaced from the position S to I (image position) by an angle θ_E, while $\tilde{r}E$ is the projected Einstein radius. The formula of the relationship between θ and the angular separation β is given by $\beta = \theta - a_d$. From the approximation of the small angle, $\tilde{a}_d = (D_s - D_l) = a_d D_s$, and from the exterior-angle theorem, $\theta_E = \frac{\tilde{r}E}{D_l} - \frac{\tilde{r}_E}{D_s}$. Thus:

$$\beta = \theta - \frac{4GM}{rEc^2}\frac{D_s - D_l}{D_s}$$

When a gravitational microlensing effect occurs and supposing that the observer, lens and source are in perfect alignment, then $\beta = 0$ and the light from the source will form a ring-like image [9]. This ring-like image is also known as the "Einstein Ring" [10] and the angular radius of this image is:

$$\theta_{E}= \frac{4\mathrm{GM}}{\mathrm{rEc}^2} \frac{D_{S-} D_l}{D_S}$$

Given $\theta_{E=} r_E / D_l$, the formula of the absolute radius of the Einstein ring can be expressed as:

$$r^2{}_E = \frac{4\mathrm{GM}}{\mathrm{c}^2} \frac{(D_{S-} D_l) D_l}{D_S}$$

Through using convenient units of solar masses ($\mathrm{M}_\odot$), Astronomical Units (AU) and kiloparsecs (kpc), we can express the Einstein radius as:

$$\frac{r_E}{AU} = 2.85 \left(\frac{\mathrm{M}}{\mathrm{M}_\odot}\right)^{1/2} \left(\frac{D_l}{\mathrm{D}_S}\right)^{1/2} \left(\frac{D_{S-} D_l}{\mathrm{kpc}}\right)^{1/2}$$

In a further step, we can define $\mathcal{U} = \beta / \theta_E$ and $\mathcal{Y} = \theta / \theta_E$ through normalising all angles on the sky by θ_E, in order to reduce the above formula to a simplified form, as shown below:

$$\mathcal{U} = \mathcal{Y} - \mathcal{Y}^{-1}$$

However, during the observation, we are unable to see the Einstein ring on the image, since the Einstein ring is very small and cannot be resolved even with a powerful telescope. What we can measure is only the amplification of the source star.

The CCD cameras can capture lots of images and measurement data on the stars on each clear night. This measurement data will form a light curve, the light curve plot changing continuously as more measurement data are obtained. Observers use the light curve plot to determine whether or not an event is caused by microlensing.

Figure 2 shows the theoretical light curve plot for a gravitational microlensing event.

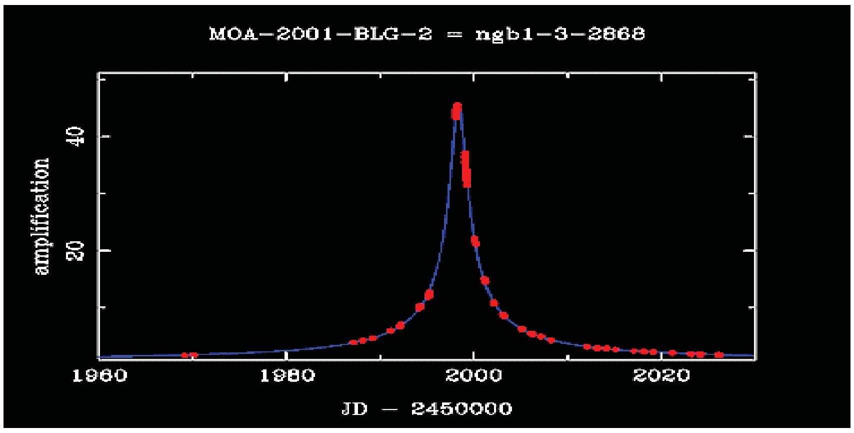


Figure 2: This microlensing light curve image is drawn from MOA 2001-BLG-2. It is a typical light curve model, with the blue line being the theoretical curve and the red dots being the measurement data [12] [13].

As Figure 2 shows, the light curve of a microlensing will rise up suddenly at a particular point, and continue to rise and amplify to another point, before quickly falling. This rising up and falling down action is called "magnification" [11]. The magnification of each image is the radius of the area of the image to the area of the source, and the formula used to calculate the amplification is:

$$A_{\pm} = \left| \frac{\mathcal{Y}_{\pm}}{\mathcal{U}} \frac{d\mathcal{Y}_{\pm}}{d\mathcal{U}} \right| = \frac{1}{2} \left[\frac{\mathcal{U}^2+2}{\mathcal{U}\sqrt{\mathcal{U}^2+4}} \pm 1 \right]$$

Therefore, the total amplification of a single lens system can be expressed as:

$$\mathrm{A}(\mathcal{U}) = \frac{\mathcal{U}^2 + 2}{\mathcal{U}\sqrt{\mathcal{U}^2 + 4}}$$

where $\mathcal{U}$ represents the separation of the source and lens in units of the Einstein radius. This radius is that of the Einstein ring when the observer, lens and source are perfectly aligned. Note that, when $\mathcal{U} = 0$, the amplification becomes infinite. However, since the size of every source star is finite, the infinite image magnification only exists in theory.

Gravitational microlensing research requires worldwide collaboration. Observers can be categorised into two groups; survey groups, and follow-up groups. The survey groups have wide field telescopes, which are capable of observing the full Galactic Bulge region and they provide initial notification of a new microlensing event to the entire microlensing research community.

On the other hand, the follow-up groups work in a different way; once they receive the initial event notification from the survey groups, they will select those events that are of interest to them and continually track the selected events in order to obtain more measurement data. This procedure is referred to as “follow-up observation”.

The first extrasolar planet that was successfully discovered via the gravitational microlensing technique occurred in event OGLE 2003-BLG-235/MOA 2003-BLG-53 (hereafter O235/M53) [9]. This event was first identified and alerted by the OGLE on 22 June 2003 [14]. However, at the time that this event was first identified and alerted, it appeared to be an unremarkable event, and none of the follow-up groups showed interest in this event, or selected it as a follow-up observation target.

One month after OGLE alerted this event MOA also detected it (on 21 July 2003), and alerted the event as MOA 2003-BLG-53. MOA observers viewed the initial light curve of this event and found that the event appeared to be in the early stages of a high magnification event. Afterwards, MOA cross checked with OGLE and found that OGLE had alerted this event one month before, so then MOA and OGLE combined their observation data to form a complete light curve and finally confirmed that this event was a planetary microlensing event.

There are many other extra-solar planets that have been observed since the first extra-solar planet was successfully discovered. As at the date of writing this thesis, there are 14 extra-solar planets that have been discovered through the use of the gravitational microlensing technique [8, 9, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33], and further extra-solar planets will be discovered through the use of gravitational microlensing in the future. Moreover, the gravitational microlensing technique is not only useful for detecting faint objects and planets in an extra-solar system; this technique can also be used to detect free floating planets, dark matter [34, 35, 36, 37], and black holes [38, 39, 40] in our galaxy system.

1.3 The MOA Project and Computational Challenges

The MOA (Microlensing Observation of Astrophysics) project is a collaborative project between New Zealand and Japan, which was established in 1995 [12, 41] at Mt John Observatory in New Zealand. It is one of the largest ongoing survey groups focusing on searching for microlensing events in both the LMC (Large Magellanic Cloud) and the GB (Galactic Bulge) fields through the use of the gravitational microlensing technique [42].

The MOA project has three wide field telescopes; a modified 0.6-m Boller & Chivens telescope, a 1.2-m telescope, and the largest, a 1.8-m optical telescope. The 1.8-m telescope is equipped with a 10 CCD chip camera, with a total of 80 MPixels. Each CCD chip has 2k X 4k pixels and it is capable of monitoring 2 square degrees of sky with a single exposure [43].

Figure 3: The 1.8-m telescope of MOA (Image source: The MOA website).

Even though MOA is one of largest microlensing survey groups and has very powerful telescopes that are able to observe millions of stars, there are, however, some computational challenges that exist for MOA.

When the MOA project detects a new microlensing event, MOA will use two classic approaches have been used to alert the new event to the microlensing research community. These are email notification, and posting the new event profile on a website.

As discussed in Section 1.2, a microlensing event has a time critical nature, so microlensing researchers want to receive the new microlensing event notification at the first time that it is detected. These two approaches are, however, not ideally suited for achieving this goal, with the drawbacks outlined below:

1. Once a new event is detected by MOA, a particular program script will read in the users' email address from a file (text format), then generate an email message that contains the new event's parameters, and then send out the message to the listed email address. In fact, using email to deliver new event notifications is not reliable, because, in some cases, email messages will have delayed delivery due to email server, or network traffic, issues. In the worst case scenario, the email message will be completely missed for some unexpected reason;

2. Lots of transient data will be generated each observation night; reading all these data with the human eye is a very complex and time consuming task. Therefore, most astronomical systems are designed as automated systems, or semi-auto systems, which require less interaction with people. When these systems receive new transient event notifications, they automatically perform some operations based on the data content (i.e. when robotic telescope software receives new transient data, it will read the data contents, selecting interesting events and automatically point to the corresponding sky position for follow up observations). In other words, the data format for holding new transient data should be able to be recognized and automatically processed by the machines. However, email is designed as a human readable format and it is not ideal for machine readability; and

3. Posting the new event profile on the website is a very inconvenient and inefficient approach, because it requires microlensing researchers to remain in front of the computer and regularly refresh the MOA webpage to check whether or not there is a new event available. Moreover, if lots of researchers refresh the website in a short time interval, lots of HTTP requests and HTTP responses will be made. Thus, the workload and the I/O traffic of the web server will significantly increase.

In fact, MOA is not the first project facing this transient event notification problem; almost all time critical projects need to resolve this problem during the system development phase. Moreover, as a gravitational microlensing event is a time critical astrophysics phenomena, when a new event has been observed, the data of the new event must be delivered to all interested microlensing researchers within one minute. This timeframe is required so that microlensing researchers can make follow up observations before the event has disappeared. Thus, the main computational challenge of microlensing event notification is developing an efficient and stable system that allows all microlensing researchers to receive new microlensing event notifications within required timeframe (one minute), with less human interaction being needed. Moreover, the system must be scalable in order to satisfy any future requirements, which is another significant challenge.

Data management is another issue that exists for MOA. In the current stage, MOA simply stores all of the observation data (more than 100 Gigabytes); including reference images, object parameters files, cross referencing files, and photometry data, among other files; in a file-based system without a data management approach having been put in place. Therefore, searching data from this huge volume of observation data has become an inefficient task. Moreover, as the volumes of data increase, the performance of data searches will decrease significantly.

The challenge of this data management issue is not so much designing a database schema and putting those observation data into the database system. The challenge is that the database solution must be sustainable, provide robust search performance and an easy to meet the future change requirements (i.e: archiving new types of data in the database). In addition, as the size of data grows very quickly, the database solution should have strong scalability and be capable of meeting the demands of storing massive volumes of data in the future.

1.4 Gaps in Computational Knowledge in the Astronomy Domain

The VOEventNet [100] is the first project that has looked for solutions to the transient event notification problem, and they developed an architecture for disseminating transient event notifications through the use of a XMPP (Extensible Messaging and Presence Protocol) [128] protocol [101]. After the first transient event notification architecture had been implemented, most of the subsequent time critical projects utilised this solution to build their own transient event notification architectures. In other words, all of the existing transient event notification architectures are XMPP based architectures. However, there are very little published literatures (only a few PowerPoint slides discuss how their architecture works) that has discussed why XMPP has been chosen for disseminating transient event notifications.

However, the XMPP is not the only solution for disseminating transient event notifications; there is another technique called Java Message Service (JMS) [238]. The JMS technique has the ability to provide the same messaging delivery features as XMPP, but it also supports lots of additional messaging features, such as durable subscription and rich message formats. The additional messaging features supported by JMS allow us to implement a powerful, extensible and scalable architecture.

The JMS has been widely used for data dissemination by projects in other domains. For example; the University of Washington has used JMS to distribute messaging to various internal applications, and the Finnish IT Center for Science has an event-based architecture that uses JMS to exchange data between the servers and clients [198]. It is surprising that JMS has not been used in the astronomy domain, but the JMS system is definitely worth considering for use as an alternative solution for disseminating transient event notifications.

Moreover, various performance benchmark studies of JMS [199] [200] [201] [202] [203] have already been carried out in other domains, but the results shown in those studies may not fit the context of a time critical astronomy domain. This is because the payload size of message used in those performance benchmark are relatively small (from several bytes to 1 kilo bytes), however, in astronomical domain, the payload size of message could be up to 1 megabyte. Moreover, data rates that need to be processed by time critical astrophysics projects are also very high, but there is no existing literature that has discussed JMS performance under a high data rate situation with large size of message payload. Thus, it is also necessary to examine whether or not the JMS architecture is capable of handling this situation.

Data management is another interesting topic in the astrophysics domain. This is because there are lots of different observational data that will be captured in each observation night, and these data need to be well archived for future analysis. Most of the existing astronomical projects use a relational database (RDBMS) for astronomical data storage, and RDBMS does a very good job in the current stage. Indeed, another type of database system has become very popular, namely NOSQL databases [204]. A lot of projects in other domains have start using NOSQL databases for their data storage (i.e. Google, Facebook, and Twitter among others [205][209]), because when the data size grows, RDBMS may encounter some scalability issues [206] [207]; in contrast, one of the highlighted features offered by NOSQL databases is strong scalability [208]. In fact, the NOSQL database has not been designed to replace the RDBMS system; it is designed to address some issues that RDBMS cannot. However, no recent studies have been conducted on the use of NOSQL databases for astronomical data management. Thus, there is an opportunity to study the feasibility of utilising a NOSQL database for astronomical data management in order to fill the gap in the astronomy domain.

1.5 Objectives of the Research and Research Questions

This research covers two topics; transient event notification architecture, and data management systems for astronomical data management. The major objectives to be accomplished in this research are:

1. Investigate the possible techniques that could be used for disseminating astronomical transient event data;

2. Propose JMS based architecture for transient event notification; this architecture should be able to meet the demands of high event rate situations. The findings of this study should be adaptable to requirements of projects in other domains.

3. Extend (2) above to develop a mobile application for receiving the transient event notifications, and analyse the capability of this method; and

4. Investigate the feasibility of using a NOSQL database for astronomical data management, explore different categories of NOSQL databases, and develop a prototype NOSQL based data management system for managing astronomical data.

Since this research contains two different topics to address different potential computational challenges in time critical domains, there are several questions that remain and need to be investigated within this study. The full list of research questions is as follows:

1. Is the JMS architecture suitable for transient event notification delivery? Is it able to handle high event rate situations?

2. What is the reusability of this architecture and what are the necessary components to re-implement this architecture in order to use it in other domains?

3. Given that mobile applications are currently very popular, is this architecture able to deliver transient event notifications to mobile platforms?

4. How can we organise astronomical data into a NOSQL database? What is the suitable NOSQL category for storing astronomical data, and does the NOSQL database support complex queries?

5. What benefits can we gain when using a NOSQL database as an astronomical data management system? What is the data access (data insertion and data querying) performance in a NOSQL database?

6. Could a NOSQL database be employed as a data management solution in future astronomical projects and projects in other domains?

The design and implementation of this new notification architecture will be discussed in Chapters 4 and 5. The second part of this research involves searching for a suitable database solution for MOA. This database solution must provide efficient data manipulation (including inserting, deleting and updating data from the database), ensure easy management of astrophysics data and allow for high performance data searching. This database solution will be discussed in Chapter 6.

1.6 Research Scope

There are several astronomical data analysis pipelines and different computational techniques are involved in this research, but not all of them are within the scope of this research. The research scope is limited to the following five points:

1. Implementing a new event based architecture for MOA and discussing the scalability of this architecture;

2. Benchmarking the data delivery performance of this new notification architecture in order to examine whether or not it is capable of delivering transient notifications within the required timeframe (in the gravitational microlensing context, each event should be delivered to recipients within one minute of publication);

3. Developing a mobile application framework for receiving transient event notifications and analysing the feasibility of using a mobile application for receiving transient event notifications;

4. Proposing the method of using a NOSQL database to manage astronomical data. The full details of the database design and query examples for lookup of particular data are also provided;

5. Benchmarking data insertion and data query performance with astronomical data provided by MOA.

All of the image analysis algorithms, the processes of capturing images and user input of the systems are excluded from this research scope.

1.7 Research Contributions

There are two outcomes that have been developed in this research, which are to propose new concepts of delivering astronomical transient notifications and astronomical data management. The details of each outcome are as described below:

1. The idea and concept of this new architecture will provide an alternative solution and expand the existed knowledge of disseminating transient event notifications in the astronomical domain. As mobile applications have become very popular, it would be a good idea to deliver transient event notifications through the mobile platforms. Thus, a mobile application framework has been designed and

developed in order to demonstrate the concept of how this new based architecture interacts with other non-desktop based components.

2. A NOSQL database offers some remarkable features, such as a dynamic data model, instant data structure modification, and strong scalability. It is typically suited to those projects in which the database structure is not well known in advance. This study will provide the concept of using a NOSQL database as an alternative data management solution for managing astronomical data, and add value to existing management knowledge in the astronomical domain.

Although this research is mainly focused on making a contribution to the astronomical domain, the outcomes of this research can also be applied to other domains' projects which require a solution for instant event delivery, or are interested in using a NOSQL database as their data management solution.

Chapter 2
Microlensing and Other Time Critical Astrophysics Projects

Since the first gravitational microlensing event was observed by Walsh, Carswell & Weymann in 1979, in the form of a double quasar [44], astronomers have expended much effort in the field of gravitational microlensing research, with many microlensing research projects having been established. In the first half of this chapter we will explore some important historical, present and future microlensing research projects, including both surveys and follow up projects. The second half of the chapter will explore some time critical astrophysics projects.

2.1 MACHO Collaboration and EROS Collaboration

Seven years after the first gravitational microlensing event was observed, Professor Paczynski (1940 - 2007) wrote a paper proposing that gravitational microlensing can be used to search for dark matter in the form of Massive Compact Halo Objects (MACHOs) in galaxy halos [45]. The earliest microlensing collaboration projects had their foundations twenty-three years ago, after Paczynski proposed this theory, many of the astronomers who had participated in dark matter research joined together and formed two collaboration projects: MACHO (Massive Compact Halo Object) [46] and EROS (Expérience pour la Recherche d'Objets Sombres) [47].

The MACHO project was a collaboration between the US and Australia, starting in 1991 and ending in 1999. It utilised a 1.3-m telescope equipped with eight CCD cameras, with each camera able to capture 2k × 2k pixel CCD images. The telescope was capable of observing a 0.5 square degree field of view, with the beamsplitter and filter features of the telescope having the ability to simultaneously image into two colours; red and blue [45] [51].

The observation directions of MACHO were towards the LMC, the Small Magellanic Cloud (SMC) and the Galactic Bulge (GB). During the project operation period, they captured 3TB raw image data (60% in LMC, 40% in GB and SMC), and discovered approximately 4 microlensing candidates in LMC fields and approximately 40-60 microlensing candidates in Galactic Bulge fields [46, 47] in each year.

The EROS (Expérience de Recherche d'Objets Sombres) project is a French-based Microlensing collaboration that started at 1990 at La Silla observatory (ESO) in Chile. The main objective of EROS is searching for dark matter in the form of MACHOS in the Magellanic Clouds through the Gravitational Microlensing technique; the objective of this project is similar to the MACHO project.

EROS comprised two observation phases; the first phase, called EROS-I, ran from 1990 to 1995, with two programmes involved in this phase. The first programme used a 40 cm telescope with 3.5 million pixel CCD cameras to search for short timescale microlensing events ranging from a few hours to a few days. This programme started in 1991 and ended in 1995, with no candidates found in this short timescale range during the programme's operational period. The second programme ran from 1990 through 1993; it used a 1m Schmidt telescope to search for microlensing events in LMC fields. This programme also discovered the first two microlensing candidates as reported by the MACHO project in 1993 [53].

In 1993, the EROS project started to build a new 1m diameter f/5 Ritchey-Chretien telescope for the second observation phase (EROS-II). The new telescope was equipped with two colour CCD cameras and includes a beam-splitting dichroic cube, allowing simultaneous imaging in two wide pass bands [50] (blue and red).This telescope was operated from 1996 until 2003, and it was specially customised for observing microlensing events. The EROS-II programme mainly focused on searching for microlensing events from the 3 different sky directions: the Magellanic Clouds (60 fields for the Large, 10 fields for the Small), the Galactic Centre (67 fields) and 4 areas in the Galactic Plane (approximately 6 fields each) [50]. Each observation direction contains more than a hundred-million stars. During the operational period of the programme, more than 100 microlensing candidates were discovered towards these 3 directions. Moreover, beside microlensing, EROS-II also conducted a semi-automated search for supernova, and discovered their first supernova phenomena in 1997 [50].

The MACHO and EROS projects are past microlensing survey projects that are no longer operating, however, these two projects contributed many important resources (including light curves, observation images and photometric data, among other developments) for use in future microlensing research.

2.2 OGLE Collaboration

The OGLE (Optical Gravitational Lensing Experiment) collaboration project is one of largest ongoing microlensing surveys. It was started in 1991 and is led by Professor Andrzej Undalski. It is designed for searching large numbers of microlensing events. The observatory used by the OGLE project is located at Las Campanas Observatory and operated by the Carnegie Institution of Washington [54, 55]. The OGLE project can be divided into four phases, with the main objectives being to search for dark matter and planets toward the Galactic Bulge, Magellanic Clouds (Large and Small) and the Galactic Disk via the use of the gravitational microlensing technique.

The first phase (OGLE-I) started in 1991, with the observation season running from 1992 through 1995; this was the project's pilot phase. In this phase, OGLE used a Swope 1m telescope equipped with a single chip 2K × 2K pixel CCD camera to monitor millions of objects and search microlensing events towards the Galactic Bulge. No results were found in the first observation season.

The first microlensing event toward the Galactic Bulge was discovered by OGLE in 1993. Their subsequent discovery paper [56] described this event as “a classical microlensing event and the shape of the light curve is nearly perfect match the theoretical light curve”. After the first microlensing event was discovered, OGLE continued its success by discovering two new microlensing events in the same observation season.

One year after that, OGLE discovered a further two new microlensing events toward the Galactic Bulge, but the most important development was the Early Warning System (EWS) in 1994. The EWS was the earliest microlensing notification system used to distribute the new microlensing candidates to worldwide astrophysics communities and was capable of almost real time analysis.

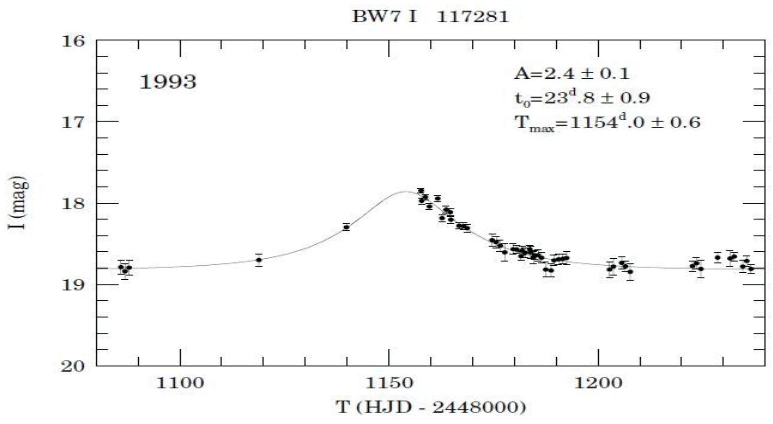


Figure 4: Light curve of the first microlensing event discovered towards the Galactic Bulge [56].

The workflows of EWS can be summarized into three steps: First, every new piece of data is automatically analysed by an on-site workstation computer; second, the on-site workstation computer selects all suspected light curves and sends these data to Warsaw University Observatory for further analysis; and finally, once the suspected light curves have been confirmed are microlensing candidates, they will be named a “prime microlens candidate” [57]. New event notifications are then sent out to interested astronomers as emails and the data for the event are displayed on a website.

In 1995, OGLE started to build the second phase of the project (OGLE-II), which started operations in the 1996 observation season. OGLE-I was officially shut down in the same year, with many important results having been found during the OGLE-I operation period; a total of 12 microlensing events were found [58], the first microlensing event was discovered towards the Galactic Bulge in 1993, the first microlensing caused by a binary object was discovered, and a new event notification system (EWS) was developed.

OGLE used the newest 1.3 m R/C (Richey/Chretien) telescope as its main observation instrument in the OGLE-II stage. The new telescope provided a full 1.5 × 1.5 degree field of view and was capable of monitoring 9 million stars in Magellanic Clouds (Large and Small), 3 million stars in the Galactic Bulge and 600,000 stars in the Galactic Disk. Moreover, the new telescope was mounted with a first generation 2K × 2K pixels Scientific Imaging Technologies (SITe) CCD camera. This was the first generation of CCD cameras to capture images in 2K × 8K pixels, with approximately 1.7-2.3 GB (Gigabytes) of raw images generated each night, and a total of 1000 GB per year [59].

With the new instrument, the microlensing event detection rate increased from 4 new events per observation season to approximately 50 new events per observation season, with more than 200 new events discovered [60] during the 4 years that OGLE-II was run.

OGLE-II was suspended at the end of the 2000 observation season, with the project stepping into its third phase (OGLE-III) in 2001. There were three major upgrades in the OGLE-III phase. The first upgrade replaced the old single chip CCD camera with a second generation 8 chips 2K by 4K pixels SITe mosaic CCD camera. This new camera was capable of covering 35 x 35 arcmin in the sky, including the entire area of the Magellanic Clouds (Small and Large) and the large field of the Galactic Bulge. Moreover, the new CCD camera also had the ability to photometer more than 100 million stars each night [61] [62]. The data rate reached approximately 1-2 TB (Terabytes) per year, which was 8 times larger than in the OGLE-II phase.

The second upgrade involved updating the EWS (Early Warning System) system; the new EWS system used a brand new algorithm for detecting brightness change in the objects [63]. With the new EWS system, the event detection rate increased significantly over the previous phases, with approximate 300 to 600 new microlensing event candidates discovered and alerted to the public in each observation season.

Observation season	New microlensing event candidates
2002	350
2003	450
2004	600
2005	550
2006	550
2007	600
2008	650

Table 1: New microlensing event candidates discovered by the new EWS system during the OGLE-III phase [64].

The last important upgrade carried out in OGLE-III was the development of two new real time data analysis systems; EWS (Early Warning System), and NOOS (New Objects in the OGLE SKY) [63]. The EWS was designed to detect anomalies of microlensing events from a single mass microlensing light curve profile in real time; the EWS also allowed observers to have fast access and response to the current behaviour of ongoing microlensing events that had previously been alerted. Furthermore, the EWS enabled observers to switch the observation mode from the normal survey mode to the follow up mode in order to allow higher temporal sampling of the events.

The objective of the other real time data analysis system (NOOS) was to detect all possible transient objects (including supernovae (SNe), long term variable stars, microlensing of very faint stars, and some other transient natural objects) in the OGLE-III sky fields. In fact, this system was very useful, because OGLE detected a few new supernovae just after NOOS was developed [65].

The OGLE-III phase ran until the end of the 2009 observation season, with a total of 2,786 variable stars, 92 multi-mode objects and 67 pulsating stars found during the

OGLE-III operating period. Important results in this phase were the detection of five extra-solar transiting systems and the contribution to the discovery of the first exoplanets via the microlensing technique [66].

OGLE started its current phase (OGLE-IV) in the 2010 observation season. The main objectives of OGLE-IV are to increase the number of microlensing events detected and search for planets in the extra-solar systems. Two major upgrades have occurred in the new phase. The first of these has been updating the CCD camera instrument; a third generation mosaic CCD camera with 256 mega-pixels has been installed on the telescope to replace the old 16 chip camera. The new camera has 32 chips, with each chip providing 2K by 4K pixels, and the new CCD camera covering a total of 1.4 square degrees field of view in the sky. With the new CCD camera, OGLE-IV will produce approximately 50TB of data per year [67], which is 10 times more than OGLE-III.

The second major upgrade is that OGLE updated its EWS system in order to satisfy the OGLE-IV data pipeline in 2011. The new EWS system [64] is very similar to the OGLE-III EWS system, but the detection rate for microlensing event candidates is about 3 times larger than that for OGLE-III. Approximately 1,700 microlensing events per year were alerted by the new EWS system in the first three observation seasons.

During its 23 years of operation, OGLE has collected very large amounts of photometry data for analysis in order to determine potential microlensing event candidates. These data are recorded in a SQL type database [68] that is publicly accessible. OGLE provides a public web interface that allows astronomers to query data from the database for study purposes. At the time of writing this thesis, a total of 17 planets have been discovered by OGLE, with six of these planets discovered through the use of gravitational microlensing techniques.

2.3 Follow up Observation Groups

As mentioned in the previous sections, gravitational microlensing is a worldwide collaboration project. The survey groups are responsible for issuing initial alerts of new transient events; the follow up groups will pick particular events for follow up observations in order to obtain more data about the events; and once the new data has been collected by the follow up groups, they will share it with the community as soon as possible for future analysis. Three existing well-known follow up groups are the MicroFUN (Microlensing Follow-Up Network) [69], MiNDSTEp (Microlensing Network for the Detection of Small Terrestrial Exoplanets [70] and ARTEMis (Automated Robotic Terrestrial Exoplanet Microlensing Search) [181].

The MiNDSTEp project started its pilot phase (also referred to as MiNDSTEP0) in 2008 with the Danish 1.5m telescope at ESO LaSilla (Chile). One year after, this project expanded to include two more telescopes in order to expand its telescope network. These two telescopes are the MONET 1.2m telescopes at McDonald Observatory (Texas, USA) and SAAO (South Africa). The main objective of this project is to study the population of planets down to Earth mass, and even below, in the Milky Way through monitoring of ongoing gravitational microlensing events [70].

ARTEMis is an expert system designed to automatically detect anomalies that have been active since 2008. It has the ability to recommend and select optimal ongoing events for follow up observation at any given time based on a priority algorithm, in order to increase the expected number of planet detections [181]. Moreover, the ARTEMis system also provides other services, such as alerts identifying ongoing anomalies, real time light curve modelling, displaying information about the nature of ongoing anomalies, and so forth. These services offer great opportunities to share new research progress, live information about ongoing events and delivery of the concept of “science live to your home” to the general public.

The MicroFUN is an informal group with the aim of observing high magnification microlensing events that offer the greatest potential for detecting extrasolar planets orbiting the lensing star in the Milky Way's Galactic Bulge. It has a network of telescopes operated at observatories in a number of countries, such as New Zealand, Australia, South Africa, the US and Chile.

These three follow up groups work very closely with the MOA and OGLE surveys and contribute greatly to microlensing research.

2.4 Next Generation Microlensing Survey Projects

The OGLE and MOA projects are two existing microlensing surveys that are currently operating. These two surveys have been very successful in discovering planets and variable objects in our galaxy via the gravitational microlensing technique over the past 20 years. In the future, however, more surveys will be coming online to join in the microlensing research. In this section we will now explore two of the most important future surveys, still under development and which will soon be coming online.

2.4.1 KMTNet (Korea Microlensing Telescope Network)

The KMT (Korea Microlensing Telescope Network) is a new generation microlensing survey that has become operation in 2015 [264]. The main goals [71] of KMT are to discover extra-solar planets towards the Galactic Bulge via the gravitational microlensing technique.

The main observation instruments of KMT are three 1.6 m wide field telescopes placed at three different sites (Chile, South Africa and Australia). Each telescope is equipped with a 4 chip mosaic CCD camera with 10K by 10K pixels, which is capable of covering a 2 by 2 square degrees field of view. Unlike existing microlensing surveys, the KMT will not require follow up observation, because the telescope networks are located in 3 different time zones areas at similar latitudes, so in theory, it is possible to continuously monitor and obtain the data of particular targets 7 days a week and 24 hours, but still needs to depending on the weather condition.

The size of the data flow of KMT will be larger than any of the existing microlensing surveys (OGLE, MOA). The raw images and photometry data that is obtained through the telescopes will be transferred to the KMT data centre for further analysis via general data transfer methods; either Internet, or airmail. In addition, the KMT uses database as a major data management method to manage all of the photometry data of variable objects obtained by the telescope network.

2.4.2 LSST (Large Synoptic Survey Telescope)

The LSST (Large Synoptic Survey Telescope) is a new generation of wide field reflecting telescope sky survey project, and will be located at El Peñón Peak in Cerro Pachón, Northern Chile. This project is still under construction and will begin full operation in 2022. It will then run for 10 years.

The main scientific goals of LSST are a mapping inventory of the solar system, searching rare transient events (such as novae, supernovae, neutron stars, stellar flares, gamma-ray bursts, X-Rays and another possible new types of transients), exploring rapid changes and moving objects (such as hazardous near-Earth asteroids and distant Kuiper Belt objects) in the solar system, searching dark energy and dark matter via weak lensing (WL) techniques, and mapping the Milky Way [72] [73].

The telescope used by LSST is a new generation 8 m ground based telescope equipped with a 3200 Megapixel camera that is able to cover 9.6 square degrees field of view; in order to cover this wide range field of view, 3 mirrors [74] [75] need to be installed on the telescope, rather than two as with current telescopes. The primary mirror of the telescope is 8.4 m in diameter, the second is 3.4 m in diameter, and the third one is 5 m in diameter and is located in the hole of the primary mirror.

The data flow rate of LSST is expected to be 5 PB (Petabytes) per year [76], with over 50PB of imaging data and more than 10 PB (Petabytes) of catalogue data recorded by the end of the 10 year observation period. The telescope will capture approximately 30 TB (Terabytes) of imaging data on each observation night; the data reduction pipeline will analyse the imaging data in pairs of 6.4 GB within 60 seconds of the imaging data being captured in order to determine transient events and then alert the entire astrophysics community. In addition, the LSST uses a 2.5 Gbps network to transfer these large amounts of data from the observatory to the data centre, which is located in the US. This data centre will allow public users to access and analyse the imaging data online without the need for them to download large subsets of data.

The LSST is a large dataset and a very time critical astrophysics project, with approximately 30 existing middleware packages and libraries involved in order to fit the need for data persistence, data transfer and retrieval, parallel processing, visualisation, data control and network security [77]. The proposed main data management system [78] is a MySQL layered database that includes more than 100 tables, 700 million rows of image metadata, 3 trillion rows of source catalogue, and 300 billion rows of object catalogue with each row of the object catalogue containing more than 200 attributes, 10 million rows of moving objects, 100 million rows of variable objects, calibration metadata and some other types of relative metadata.

All of these computational systems are still under development, with an expected completion at the end of 2018. The LSST will also provide a set of application products that allows end user download and use for data retrieval, data access and data analysis after the project has started operating. The LSST project provides the opportunity for astronomers to study, explore and map the details of our universe.

2.5 Other Time Critical Astrophysics Projects

Some astrophysics projects are interested in observing and researching other astronomical phenomena with time critical natures, other than gravitational microlensing.

These phenomena include blazars, supernovae and hypernovae as well as other types of transient objects. These projects will be reviewed in this section.

2.5.1 CSS (Catalina Sky Survey)

The Catalina Sky Survey (CSS) has the aim of mapping the inventory of near earth objects (NEOs) and potentially hazardous asteroids (PHAs), which may pose potential risks to Earth. The original CSS [79] began with a prototype stage in 1998, operated by Steward Observatory of the University of Arizona.

The initial telescope used by the original CSS was a 0.4/0.6 m Catalina Schmidt telescope. In September 1999, CSS added automated detection software to the telescope and continued to operate until April 2000. During those seven months of operation, CSS discovered 46 new NEOs and made 175,000 astrometric measurements.

After the prototype stage, the CSS was temporarily shut down in order to upgrade the telescope and the sub-software systems. The project officially resumed operation in November 2003, with a new 0.7m f/1.9 Catalina Schmidt telescope equipped with a single unfiltered 4K by 4K CCD camera able to cover an 8 square degrees field of view as the main observation instrument. With this new system, the discovery rate was significantly increased over the prototype stage.

In 2009, the original CSS collaborated with another survey; the Mt. Lemmon Survey (MLS); to form a new CSS in order to search for more NEOs and PHAs. The Mt. Lemmon Survey was operated by the Steward Observatory of the University of Arizona, located at Mount Lemmon in the Catalina Mountains north of Turcson. It uses a 1.5 m Cassegrain reflector with a CCD mosaic camera that is capable of covering a 1 square degree field of view as the observation instrument. The main scientific goals of Mt. Lemmon are to make follow up observations of near-earth asteroids and minor planets in the solar system, gathering accurate time series photometry data to determine the size, shape and other attributes of these objects [79].

The CSS has successfully discovered more than a hundred PHAs, comets and other objects [80] in our solar system.

2.5.2 CRTS (Catalina Real Time Survey)

The Catalina Real Time Survey (CRTS) [81] [82] aims to discover varying types of transients (such as supernovae, cataclysmic variables (CV), blazars, active galactic nuclei and flares), and distribute all of these new discoveries to the public in real time without a data proprietary period, so that all astronomers can make follow up observations on particular ongoing transients immediately and without any delays.

The dataset used by the CRTS is based on the data capture from the CSS telescopes [83], and the data processes pipeline is fully automatic. When new data is obtained by the CSS telescopes, the CRTS will analyse those data to search for new transients via machine learning techniques; any newly detected transients will be fed to the data streams on two affiliated project platforms; VOEventNet and SkyAlert (these two projects will be discussed in the following chapter); within 60 seconds through a new Internet-based mechanism, rather than using a traditional email-based data distribution mechanism. The new transient profiles are formed as VOEvent formats when transferred over the network; this format is defined by the IVOA (International Virtual Observatory Alliance) as a standard transient alert format, and VOEvent format will also be introduced in the following chapter.

The CRTS has implemented *open data* theory and has opened a new era of data collaboration in the time domain astrophysics field. As discussed by Drake [84], "open data shift focus from the ownership of data to the ownership of expertise". In addition, another benefit brought by this project is the increase in scientific results from new transients' discoveries.

To date, over 1200 unique transients, over 300 SNe, over 300 CVs, over 250 blazars, approximately 50 flarings and other high amplitude variable stars with unknown natures [83] have been successful discovered by the CRTS.

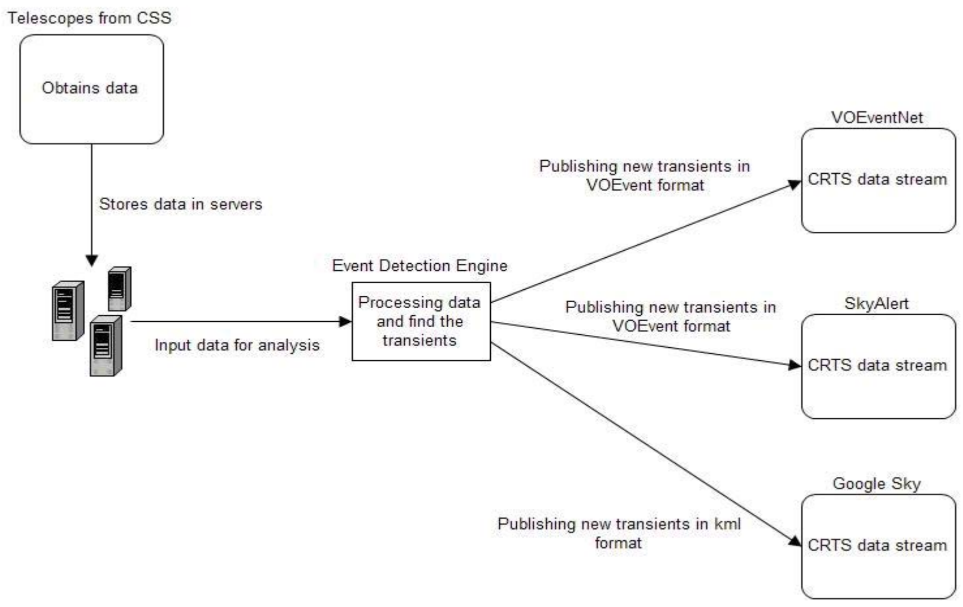


Figure 5: Data processes flow of CRTS.

2.5.3 WFIRST (Wide Field Infrared Survey Telescope)

The Wide Field Infrared Survey Telescope (WFIRST) is a NASA (National Aeronautics and Space Administration) observatory, which is designed to capture wide field imaging and slitless spectroscopic surveys of the near infrared (NIR) sky for use by the entire astrophysics community [85]. The current high priority mission of WFIRST is to implement the Astrophysics Focused Telescope Assets (AFTA), originally proposed by NASA astronomers in 2003. The AFTA is a new generation space mission that focuses on searching the dark energy via weak gravitational lensing and searching the exoplanets via gravitational microlensing in order to resolve the questions remaining in both of these research areas. In addition, the AFTA includes a guest investigator mode, which provides the ability to conduct survey investigations of nearby galaxies to study and gain an understanding of their formation and structure [86].

The AFTA project uses existing spacecraft, ground system architecture, hardware and a 2.4 metre wide field telescope as the main observation instruments; the telescope is equipped with a coronagraph instrument and is able to undertake direct imaging of extrasolar planets and the debris disk. This project is still in a pre-formulation phase, which will be completed in early 2016. After this pre-formulation phase is completed [87], the baseline design will start in 2017, with 7 years of design period. The currently planned launch date of this project is scheduled for 2024.

2.6 Conclusions

In this chapter, we explored past, current and future projects related to gravitational microlensing research and time critical astrophysics research. These projects are, however, all based on astrophysics aspects; the astrophysics projects that are based on computational aspects and the relative techniques that make it possible to use them in implementing a time critical astrophysics system will be introduced in the next chapter.

Chapter 3 Computational Oriented Time Critical Astrophysics Projects and Relative Techniques for Implementing Real Time Transient Notification Architecture

Time critical astrophysics projects can be categorised into two different classes. The first class focuses on astronomical aspects, which are surveys and follow up groups, as introduced in the previous chapter. The other one focuses on computational aspects, which includes work on designing standard frameworks to represent astronomical transients' data, implementing computational architecture that has the ability to rapidly distribute new transients to the public astronomy community, and implementing application products for use by end users in retrieving new transients in a convenient manner.

The first two sections of this chapter discusses two standard astronomical formats that are widely used for holding and representing the metadata of transients, and two computational projects that distribute new transient notifications to the public in real time. The remainder of this chapter discusses related techniques that have the ability to provide real time data delivery.

3.1 IVOA Standards

Time critical astrophysics research is worldwide collaborative research, so there are many different surveys with the aim of researching and observing different types of transients; each survey has its own data centre, which stores all of the observation data obtained by its telescope, and these observation data have accessed by global astronomers through astronomical analysis tools.

Moreover, the surveys will also initiate sharing their data with the globally distributed astronomical community and set up systems for alliances to carry out analysis of the data (i.e., publishing new microlensing data to a worldwide follow up network).

As mentioned in the previous chapter, each survey will generate large amounts of astronomical data, and each survey has its own data management approach. Therefore, it is necessary to have some sort of approach that enables worldwide astronomers and astronomical analysis tools, to access and read these data in a transparent manner, regardless of how the data are stored in the data centre. In other words, the astronomical data from different data centres are inter-operable, present the same “look and feel” to astronomers, can be recognised by different type of astronomical analysis tools and work as a seamless system. Just as in the concept of the World Wide Web (WWW), a HTML (Hyper Text Markup Language) [210] document provides data from different organizations, with each document including different information, but which can be understood by all web developers and recognised by every web browser.

The key to accomplishing the goal of data interoperability between different data centres are standardisation of the astronomical data, which creates a standard framework for data representation. The concept of a Virtual Observatory (VO) is typically designed for storage and exchange of the astronomical data over the network and will be introduced in this section.

3.1.1 IVOA (International Virtual Observatory Alliance)

The International Virtual Observatory Alliance (IVOA) [88] is a worldwide astronomical organization established in 2002. It is an organization which is analogous to W3C for the WWW and the IETF for the Internet. The main task of IVOA is defining standards for astronomical data and resources obtained by different observatories. This is in order to facilitate the international coordination and collaboration necessary for development of astronomical system, computational architecture and the deployment of the analysis tools. These tools would comprise data accessing software necessary to enable the international utilisation of astronomical archives as an integrated and interoperating virtual observatory (VO) [89].

In order to achieve this task, the IVOA has proposed an architecture [90] that involves three layers called the Resource Layer, the User Layer and the Virtual Observatory Layer. The VO is an important layer within the architecture, as it connects the Resource Layer to the User Layer so that global astronomers or astronomical software are able to access the astronomical data and resources from multiple data centres via the Internet in a seamless and transparent way.

To date, IVOA has 14 members worldwide [91] [92], with some of the well-known virtual observatories being China-VO, ASVO (All Sky Virtual Observatory), Astro-Grid, VObs.it, EURO-VO (European Virtual Observatory). These projects are very successful at delivering astronomical data and services to worldwide astrophysics communities via the VO concept.

3.1.2 VO (Virtual Observatory)

The Virtual Observatory (VO) [93] is not a physical system or software product; it is a technical framework that connects astronomical resource providers (data centres) and resource consumers (individual researchers, computer systems, application software) in a seamless and transparent manner. This enables the worldwide astronomical community to access distributed astronomical resources in a single transparent system regardless of where and how those resources are stored by the resource providers.

The VO uses two concepts to achieve the goals described above. These are “resource” and “service” [94]. Resources can be anything that is able to be accessed via the Internet; for example, data collections (images, spectra, catalogues, theoretical models, and light curve data), astronomical databases, and astronomical Websites. Similarly, services can be anything that does something or provides something to the resource consumers. Data service is a good example. When the resource consumer sends a query request to the data service, the data service will query the remote astronomical resources (databases, storage spaces) and return the results to the resource consumer. In order to make these two concepts useful to resource providers and consumers, the data models are standardized in order to describe the contents of astronomical resources. The resource consumer can then easily access the different types of astronomical services and resources as necessary. As with Web context, HTML, XML [211] and JSON [212] documents are standard data models for describing the contents and metadata of resources. HTTP [213], SOAP [214] and REST [215] are protocols used to access different types of services and resources via the Internet.

Therefore, the VO framework has defined a set of standards based on top of Internet standards. This set of standards includes standardization of astronomical data and metadata (resources), standardization of exchange methods of astronomical data (protocols) and the use of a registry to list all available astronomical services. Furthermore, the standardization can be divided into five classifications; Data Access Layer, Data Modelling, Resources Registries, Semantics, and VO Query Language [91].

The Data Access Layer contains different protocols that specify basic access methods for different types of astronomical resources; for example, resource consumers can use Simple Image Access (SIA) [216], Simple Spectral Access (SSA) [217] and Simple Cone Search (SCS) [218] protocols with provide parameters (RA, Dec) in data access requests to access image archives, spectra and source catalogues in a “simple” way [91]. If the resource consumers want to access the astronomical resources in a more complex and flexible manner, they can use Table Access Protocol (TAP) instead of the simple access protocols. In addition, the access results are normally in a table format, and the VO data services are normally returned in a standardized data format called “VOTable” (introduced in Section 3.1.3) to resource consumers

The Data Modelling is designed with the intention of providing a standard framework and data structure to describe the metadata and contents of different types of astronomical data; for example, the “Phot DM” is used for describing the photometry data, the

"Obscore DM" is used for describing collections of observational data by the databases, and the "VO Event" (introduced in Section 3.1.4) is used for transient event notification. With data model standardization, any type of astronomical data is able to be reused by any applications which understand the VO data model structure.

The goal of the Resources Registries are to function as a "yellow pages" of astronomical services; they provides resource providers with the ability to place their services in this directory, so that resource consumers are able to search for the services they want to use from the registries list. Moreover, the registry is also distributed along with the resources and services, and its replica exists around the network, both for redundancy and for more specialised collections.

The aim of Semantics is to produce new standards that aid the interoperability of the VO framework. This implements standard descriptions of astrophysical objects, data types, concepts, events, and other astronomical phenomena. For example, if a metadata item is associated with a Right Ascension of an astronomical event, any astronomers or VO aware applications will understand this meaning.

Most modern distributed astronomical databases are based on SQL and allow the resource consumers to query the astronomical resource from the databases via SQL syntax. However, SQL is not the only technique that could be used for querying astronomical data. The VO Query Language (VOQL) is other technique designed for fundamental astronomical data searches; it is highly customised as the standard query language used to query any kind of astronomical data from databases within the VO framework. This query language is able to construct a query string to search the complex astronomical data in both a high level and a low level manner.

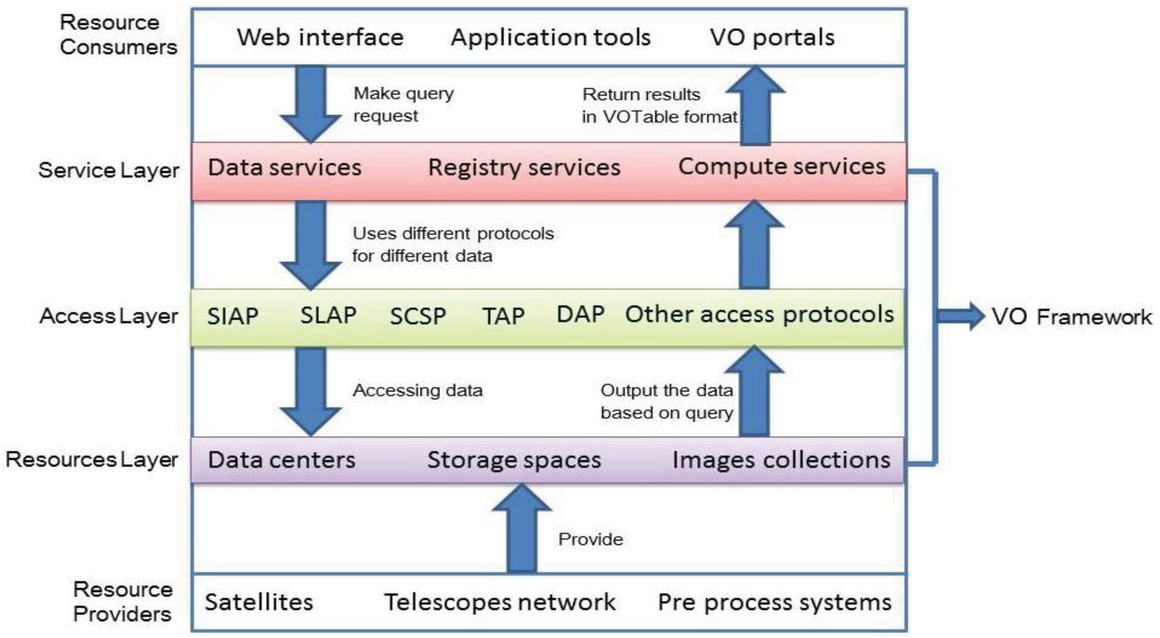
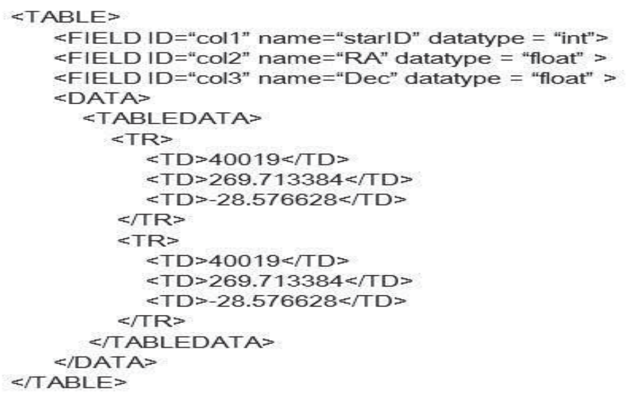


Figure 6: Processing flow of Virtual Observatory (VO) framework.

3.1.3 VOTable

The VOTable [95] is one of the most important standards in the VO context; it is widely used in many VO areas, especially in the results format returned to resource consumers by the VO data services. It is formed as an XML (eXtensible Markup Language) document and is typically designed to achieve two goals. The first goal is that it works as a flexible storage and exchange format for tabular data, especially astronomical data. The second goal is that it remains close to the FITS (Flexible Image Transport System) binary table format, so any FITS tables are able to be readily converted to VOTable format and vice versa. The FITS format is normally used as the standard digital image format in astronomical field, and the origin metadata of the image is stored as a table in the header of the FITS format.

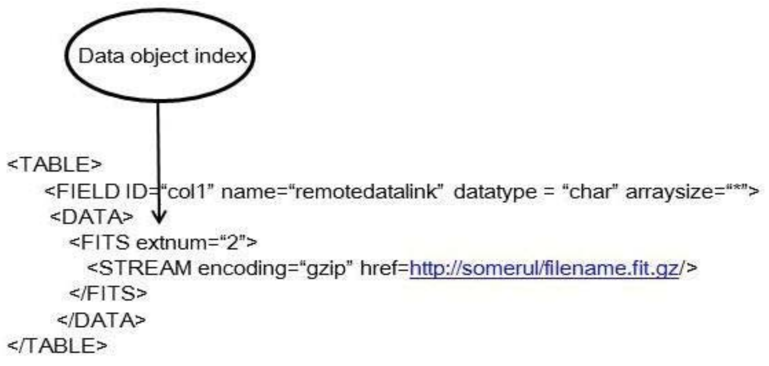
The VOTable allows the data part to be represented in three different tabular formats, which are TABLEDATA, FITS and BINARY [95]. TABLEDATA is a pure XML format so that, in this format, small tables can be easily handled in their entirety by XML tools. Each table has an unordered set of rows with data, each row in the table is a sequence of cells, and each cell contains either a primitive data type (string, double, int and so on) or an array of data with primitive data types.



```
<TABLE>
    <FIELD ID="col1" name="starID" datatype = "int">
    <FIELD ID="col2" name="RA" datatype = "float" >
    <FIELD ID="col3" name="Dec" datatype = "float" >
    <DATA>
      <TABLEDATA>
        <TR>
          <TD>40019</TD>
          <TD>269.713384</TD>
          <TD>-28.576628</TD>
        </TR>
        <TR>
          <TD>40019</TD>
          <TD>269.713384</TD>
          <TD>-28.576628</TD>
        </TR>
      </TABLEDATA>
    </DATA>
</TABLE>
```

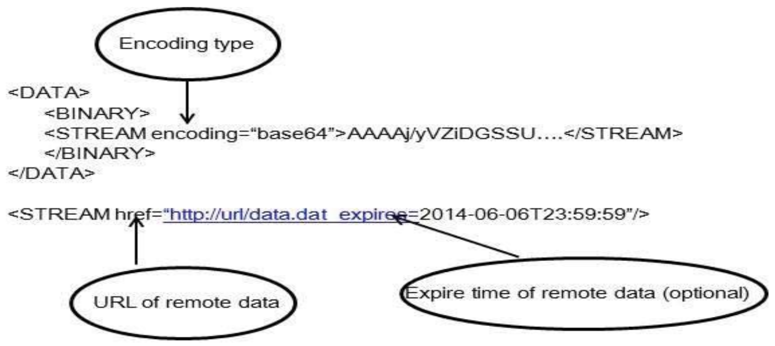
Figure 7: Example of using TABLEDATA in VOTable.

The FITS binary table format is well-known to astronomers, and VOTable can be used either to encapsulate a FITS formatted file, or to re-encode the metadata. Unfortunately, it is difficult to stream FITS, since the dataset size is required in the header (NAXIS2 keyword) and FITS requires a specification up front regarding the maximum size of its variable-length arrays. The BINARY format is supported for efficiency and ease of programming, so it doesn't require a FITS library and the streaming paradigm is supported by this format.



```
<TABLE>
   <FIELD ID="col1" name="remotedatalink" datatype = "char" arraysize="*">
    <DATA>
      <FITS extnum="2">
        <STREAM encoding="gzip" href=http://somerul/filename.fit.gz/>
      </FITS>
    </DATA>
</TABLE>
```

Figure 8: Example of using FITS in VOTable.



```
<DATA>
    <BINARY>
    <STREAM encoding="base64">AAAAj/yVZiDGSSU….</STREAM>
    </BINARY>
</DATA>

<STREAM href="http://url/data.dat  expires=2014-06-06T23:59:59"/>
```

Figure 9: Example of using BINARY in VOTable.

The key reasons that VO uses XML as the major format for storing and exchanging the astronomical data in the distributed computing environment rather than FITS are listed below:

1. The astronomical table may contain large amounts of data. Transferring all of these data over the network will reduce efficiency, however, the XML format allows the metadata and data to be stored separately, with only the reference URL of the remote data described in the XML document. So, the remote data will not be transferred until specifically requested to do so by the application. This feature is not supported in FITS format;

2. The data stored in XML format can be easily transported across two different kinds of systems. It does not matter whether the system is compatible or not, therefore, the data can be read or manipulated in different systems;

3. XML is a machine readable description and is platform independent, which means that it can be recognized by all machines and applications regardless of what platforms are being used or what program language implements the applications;

4. The FITS format requires a specific library to read the tabular data in its header file [96], but XML does not. Any data stored in an XML document can be processed perfectly by all program languages, because all program languages

have provided XML parsers to developers to develop the applications to transfer the XML data to human readable form;

5. The data structure in XML is validated strictly. The IVOA has defined an XML schema for VOTable. This XML schema imposes constraints on the structure and content of the VOTable, including what data types are allowed for each element, how each element should be ordered, the maximum number of times that particular elements can appear in each VOTable document, and so on. Developers need to follow the schema to create a valid VOTable;

6. Using the XML schema for VOTable can avoid data integrity problems when transporting data between two applications.

The VOTable has been used as one of the standard data distribution format by many virtual observatory (VO) projects [224], such as MOA, SkyAlert and Catalina Real-Time Transient Sky Survey [225][226], and IVOA still makes improvements for it based on feedback from users.

3.1.4 VOEvent

The VOEvent [97] is another important standard within the VO framework. The VOEvent standard is designed for transient notification and to provide a general enough method to describe the metadata of new transients, driving robotic telescopes for follow up observations, and triggering archive searches based on the metadata of the new transients [98].

Similar to the VOTable, the VOEvent is also based on an XML format and has an XML schema to constrain data structure. The XML schema defines the total number of each element and each attribute that can appear in a single VOEvent document. The VOEvent shares most of its element tags with VOTable, but there are six element tags that are only designed for VOEvent documents and are not included in VOTable documents; these are, <Who>, <Wherewhen>, <What>, <How>, <Why>, and <Citations>. These element tags describe the basic information of the new transients. Each tag will be explained subsequently. Any valid VOEvent document may contain zero, or only one of the six element tags [99]. The figure 10 shows components of a VOEvent document.

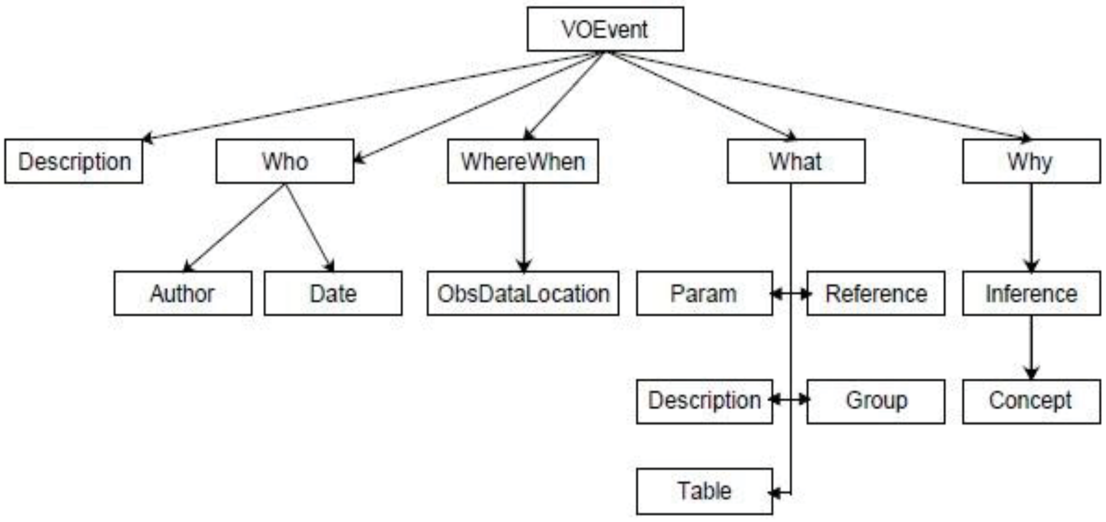


Figure 10: Components of a VOEvent document.

The element <Who> describes the source of information of the observation. This element tag may include the nest element tags <Author>, <Date> and <Publisher>. The tag <Author> describes who and which observatory has created and alerted the VOEvent packet, <Date> contains information about the publication date, and the <Publisher> tag only appears in particular VOEvent packets and is mainly used for describing the numerous logistics associated with distributing the VOEvent packet.

The <WhereWhen> element has two major purposes, as the name of the element suggests it is used to indicate where and when the transient event has been observed and sky coordinates of the transient event. More generally, it provides targeting coordination for robotic telescopes or astronomers to use when performing follow up observation of the transient event.

The `<What>` element is used for describing the actual details of the transient, such as a short description of the transient, imaging details (field, ccd chip, sequence number, and so on) and ccd position (ccdx, ccdy). Moreover, the nest element tag `<Reference>` provides the ability to link with external sources, such as the external photometry data, external finder chart image and other external datasets within the VOEvent packet. This element tag ensures the network transport efficiency of each VOEvent packet; as only metadata of the transients will be carried by the VOEvent packet, the size of each VOEvent packet becomes very small and easily consumed by the receivers, and the actual data of the transients will only be transferred when the request is made by the receiver.

The `<How>` element contains information of the observation instruments and the VOEvent also allows linking with external RTML (Remote Telescope Markup Language) documents through using a list of `<Reference>` elements that nest in the `<How>` element.

The observation history of the transient phenomena can be recorded by using the `<Why>` element in a VOEvent packet. The `<Why>` element is also associated with an attribute called *importance*; the *importance* attribute accepts values ranging between 0.0 and 1.0, which are used to indicate how important the VOEvent alert is, such that the smaller value it has, the lower the priority of the alert, and vice versa. Within the `<Why>` element are a list of `<Concept>` and `<Name>` elements associated with the nature of the transient phenomena (i.e., microlensing, CV, or other objects). These may be grouped using the `<Inference>` element, which has attributes called *probability* that are used for asserting a probability that the inference is correct and associating a description of the relation between the VOEvent alert and the `<Concept>` or `<Name>` elements within the document [99]. The value of *probability* attributes and the nature of the transient phenomena within the `<Concept>` or `<Name>` elements will change frequently as more data is obtained.

The last important element `<Citations>` is not necessary in every VOEvent packet. It allows each new VOEvent packet to refer to the previous VOEvent documents through the `<EventIVORN>` element; each `<EventIVORN>` element must be associated with a *cite* attribute, wherein the *cite* attribute describes the nature of the VOEvent packet (either it is cited or it supersedes the previous VOEvent packet) [99]. If the current VOEvent packet is superseding a previous VOEvent document, then chains of follow ups can be split or merged should it later be discovered that they describe different or rather identical phenomena.

The VOEvent document is a useful facility, tool and major format for transporting the metadata of transients to the entire astronomical community, so that the astronomers, robotic telescopes and astronomical computer systems can then decide whether the new transients require urgent follow up observation based on the information within the VOEvent packet.

3.1.5 Real Time Transient Notification Projects

As mentioned earlier, information on new transient events needs to be notified to all interested recipients as soon as it is detected. Therefore, some of the projects are focused on implementing the computational architecture in order to providing real rime transients' notification to all interested recipients.

The GCN (Gamma-ray Coordinates Network) [107] [108] is a first system that utilizes the real time notification concept for transient data delivered, it is designed by the NASA scientific group, and consists of three components; Notices, GCN Circulars, and Reports. The main objective of the GCN system is to receive and distribute information about the GRBs and other transients to interested recipients (robotic telescopes, human operated telescopes, amateurs and professional astronomers) in an automatic manner.

The Notices [109] component is typically designed for the needs of real time observations. It collects the location information of GRBs that are detected by various spacecraft or instruments, and then automatically distributes this information to all interested parties via either socket connections or email notification. The data flow of the Notices feature can be divided into the following steps:

1. The users register a New Site on the GCN website, providing all of the necessary information to receive the GRB location information from the Notices; such as an email address, socket IP address, filter condition, and what type of Notice the users wish to receive. The Site information will be stored in the GCN system once a New Site is successfully built;

2. When the GRB triggers the detector instrument on the spacecraft, the location information for the GRB will be sent down to the ground station. The ground station will then transfer the GRB location to the GCN at GFSC (NASA Goddard Space Flight Center).

3. The GCN will process the GRB location into a standard format using custom hardware and software. Then the Notices component uses either sockets or emails to automatically send a notification about the post process of the GRB location to specific Sites or emails based on a variety of pre-defined filtering conditions.

The GCN Circulars [115] collect timely information about optical, radio, x-ray and gamma-ray from the follow up community and sends this information to all interested recipients. The process flow of Circulars is much simpler than that of Notices. The follow up observers send their Circulars to a central location via email; the Circulars are then automatically forwarded to all recipients on the email distribution list in the form of a prose-style message. In addition, the email distribution list in the Circulars component is completely separate from the list in the Notices component, because some of the recipients (users and systems) might only be interested in the follow up reports, rather than the location of GRBs.

The Reports [116] portion distributes the final reports of each burst to the entire community via email. The final report is a mini paper describing the full details of the observation of each burst; this report is distributed later than the Circulars portion, giving the observers enough time to do a full analysis and make corrections to the measurements on each burst. Furthermore, the Reports portion uses the same email message style (prose-style) and distribution list as the Circulars portion.

Spacecrafts	Instruments
High Energy Transient Explorer (HETE)	Wide-Field X-ray Monitor (WXM) Soft X-ray Camera (SXC)
International Gamma-Ray Astrophysics Laboratory (INTEGRAL)	IBIS (Imager on-Board the INTEGRAL Satellite) ISGRI (Integral Soft Gamma-Ray Imager) PICsIT (Pixellated Caesium-lodide Telescope) SPI (Spectrometer for INTEGRAL)
Swift Gamma-Ray Burst Mission (Swift)	Burst Alert Telescope (BAT) X-ray Telescope (XRT) Ultraviolet/Optical Telescope (UVOT)
Astro-Rivelatore Gamma a Immagini Leggero (AGILE)	Gamma Ray Imaging Detector (GRID) SuperAGILE (SA) hard X-ray monitor Mini-Calorimeter (MCAL) Anti-Coincidence System (AC)
Fermi Gamma-ray Space Telescope (FGST)	Large Area Telescope (LAT) Gamma-ray Burst Monitor (GBM)

Table 2: Spacecrafts and onboard instrumentation that detects and provides location information of GRBs to the GCN system [108] [110] [111] [112] [113] [114].

Gamma ray bursts are a very short duration transient phenomenon, and the duration of each gamma ray burst ranges from a few seconds to a few minutes [218][219][220][221], which is much shorter than a gravitational microlensing event. Thus, the GCN system reduces the time delay for delivery of the location information about GRBs from the spacecraft instruments to the follow up observation telescope. When a new GRB event has been detected, the GCN system can provide a notification regarding the coordination of this new GRB event to follow up telescopes and observers within a few seconds [222]; this means that follow up telescopes and observers are able to makes follow up observations while the GRB is still bursting. In addition, the GCN system also allows the GRB follow up parties to make maximum use of limited resources (labour and telescope time) by communicating what has already been done, or what is going to be done soon.

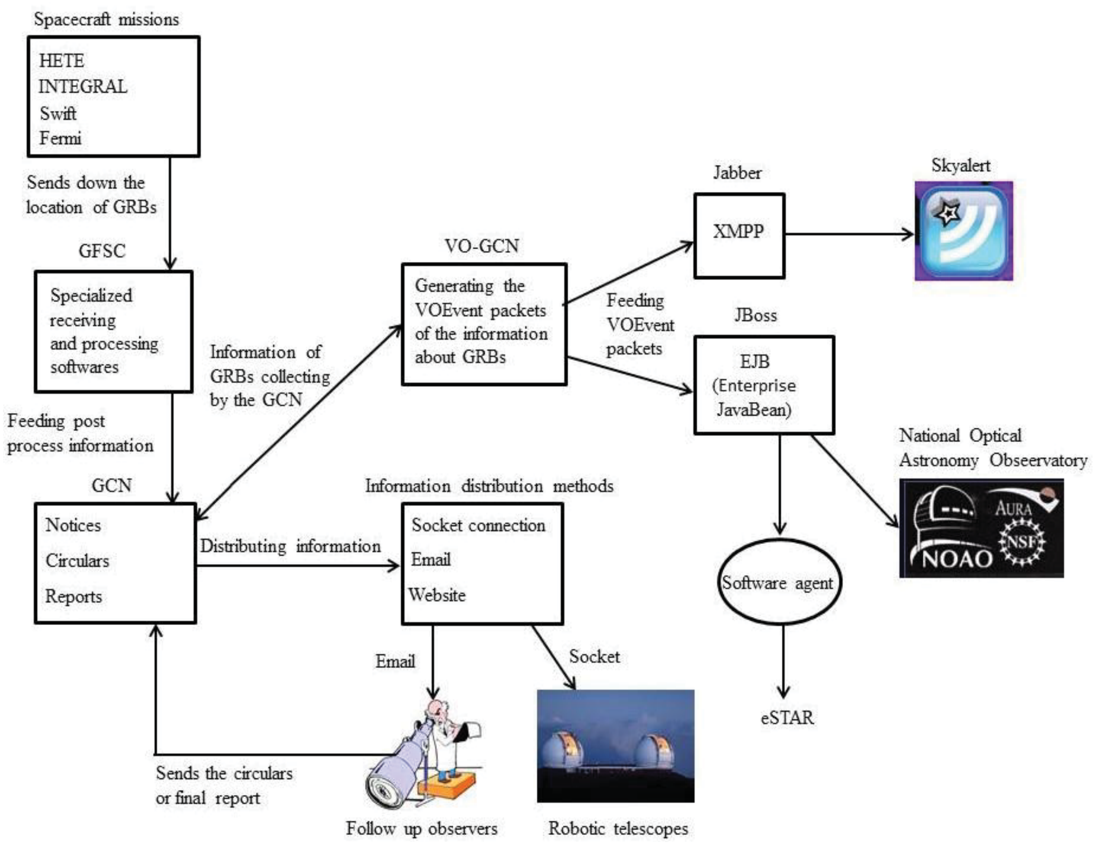

Spacecraft missions
HETE
INTEGRAL
Swift
Fermi
Sends down the location of GRBs
GFSC
Specialized receiving and processing softwares
Feeding post process information
GCN
Notices
Circulars
Reports
Information of GRBs collecting by the GCN
VO-GCN
Generating the VOEvent packets of the information about GRBs
Feeding VOEvent packets
Jabber
XMPP
Skyalert
JBoss
EJB (Enterprise JavaBean)
National Optical Astronomy Obseervatory
AURA
NSF
NOAO
Software agent
eSTAR
Distributing information
Information distribution methods
Socket connection
Email
Website
Email
Socket
Sends the circulars or final report
Follow up observers
Robotic telescopes

Figure 11: The overview and processing flows of the GCN architecture.

The GCN system also cooperates with VO projects (such as SkyAlert, NOAO (National Optical Astronomy Observatory [117]), and eStar) to form a system called the VO-GCN VOEvent Network. The VO-GCN system uses information on GRBs, or any other transients that are collected by the GCN, and processes this information into VOEvent packets. Once the VOEvent packet has been created, the VO-GCN system will distribute the new VOEvent packet to two different types of backbone servers (Jabber and JBOSS); each cooperating VO project has a software agent connected to one of these two backbone servers. When the new VOEvent packet is available in the backbone server, the backbone server will forward this new VOEvent packet to all online software agents via the Pub/Sub service [117] [223].

The GCN system reduces the time delay for delivery of the location information about GRBs from the spacecraft instruments to the follow up observation telescope. It also allows the GRB follow up parties to make maximum use of limited resources (labour and telescope time), by communicating what has already been done or is going to be done soon.

The GCN system now has more than 600 sites, and has successfully made more than 7,500 observations on around 1,300 GRBs, with a system up time of more than 98% over its 19 years of operation [118].

The VOEventNet [100][101] project is another typical example of a real time transient notification architecture project that aims to transport VOEvent packets from surveys to all interested astronomers, robotic telescopes and other astronomical computer systems in near real time. The VOEventNet project has three major concepts; broker, publisher, and subscriber. The broker is a server that maintains a list of event streams (each event stream represents a single survey), receives VOEvent packets from publishers and broadcasts to all subscribers. The broker also has the ability to know what type of VOEvent packet each subscriber wants to receive.

The surveys that fill the role of publisher are responsible for generating and publishing the VOEvent packet to the broker; each survey will publish their own VOEvent packets to the corresponding event stream on the broker.

The astronomers, robotic telescopes and astronomical computer systems fill the role of subscribers. They establish a long term connection with the broker, subscribing to the event streams which interest them and waiting for new incoming VOEvent packets.

The VOEventNet architecture is built upon these three concepts and all transient event streams on the broker are publicly available. Moreover, the VOEventNet also implements a number of open source software clients for publication, subscription and reception of VOEvent packets to and from the broker; these clients are implemented based on mainstream program languages and can be download from the VOEventNet website.

After operating for a few years, the VOEventNet project has been replaced by another project called SkyAlert [104]. SkyAlert is a clearing house and repository of information about astronomical transients; each transient is described by a collection of VOEvent packets; people and machines can access information on recent and past transients through using either a Web page interface or a Web service interface. In fact, the SkyAlert project not only aims to provide a way to browse the historical information on transients, but it also aims to provide a system that can work with all transients in general. The advantages of a general transient system are increased interoperability and the avoidance of duplicated efforts in implementing similar systems. The core features of the SkyAlert system [105] [106] are:

- It acts as a web author publishing system. The survey's role as author is that it is responsible for publishing information on newly discovered transients. However, SkyAlert requires that each author (survey) creates a sample event at the initial stage; the sample event must contain all of the necessary parameters and full metadata that might be present in the real event. After the sample event has been created, SkyAlert will create a new event stream on its server for the author. The author can use either the web form or the web service interface to publish information on newly discovered transients once the new event stream is online;

- A web subscription and broadcast system. This is a list of event streams that are available for public subscription. Subscribers can obtain information on all new transients through the event stream they choose. Moreover, the system also allows subscribers to create custom feeds providing a Python expression on the values of parameters, so that only those transients that satisfy the expressions will be delivered to the subscribers, rather than delivering every new transient to the subscribers;

- It is a repository that stores all published transients, and it also allows users to query particular events on the web form;

Stream Name	Belong to Project
AAVSO	AAVSI Alerts and Special Notices
CRTS, CRTS2, CRTS3	Catalina Real time Transient Survey
CSS_NEO	Catalina Sky Survey (Moving objects)
MOA	MOA Microlensing Survey
OGLE	OGLE Microlensing Survey
SWIFT	SWIFT GRB alerts
HST_MCT	CANDELS or CLASH (Supernova)

Table 3: Some available event streams on SkyAlert, the full list of event streams can found at the SkyAlert website [104].

- A multi-layer web page for browsing recent and past information of transients. Each transient has its own web page called a portfolio. The portfolio is a new concept invented by SkyAlert, and it is a collection of information about the transient, including the finder chart, event parameters and the VOEvent packet. The event portfolio also inherits the citation mechanism from the VOEvent packet. If an event cites another event, it will join the portfolio to which the other transient belongs. Otherwise, an event will have its own portfolio page. Moreover, the event portfolio page provides three different options to users browsing information about the transient; an overview, the parameters, and XML. The overview shows the finder chart, the light curve of the selected transient and photometry data of the transient, which is also provided through a clickable link. The `params` presents all parameters of the transients in a tabular format, while the XML option shows the actual VOEvent packet that was received by SkyAlert; and

- An open development platform allows for the implementation of a local client application, as well as the web based application.

The SkyAlert aggregates different sources of transient events into a single platform, but it is not only a repository system; it is also a transient notification system that allows surveys to share new information of transients with worldwide astronomers, amateur astronomers and telescopes in almost real time. This real time sharing feature is very important in the context of time critical astronomical research.

The concepts and protocols that SkyAlert, VOEventNet and GCN use are not the only way to providing real time transient notification. In the remaining sections of this chapter, we will discuss some concepts and protocols that are relevant in implementing a real time transient notification architecture.

3.2 Polling Mechanism

The polling approach is one possible way to implement a real time transient notification architecture. This approach is based on the request/response messaging pattern and is widely used for exchanging data between distant entities in both computer science and web contexts. The polling approach is an N to 1 model, where N is the number of clients and 1 is the server. The clients request information updates from the server, and then the server responds to the clients.

The term "polling" is a general designation for this approach; it can be divided into two types: the short polling approach and the long polling approach. The aims of these two types are the same, which is retrieving new information from the server. The implementation of work flows between these two types is, however, different. The full details of each type are explained in the following sub sections.

3.2.1 Short Polling Approach

Short polling [119] is also referred to as synchronous polling. The client side initiates the connection and requests update information from the server periodically at predefined intervals. When the client makes handshakes with the server successfully, the server will answer the client immediately regardless of whether or not new information is available on the server side.

This approach has a big drawback [120] [121]; whenever new information is available on the server side, the new information can only be notified to the client when the next polling cycle is triggered by the client. In other words, the new information cannot be updated to the clients in real time. This drawback can be solved by reducing the interval time; for example, the polling interval can be set down to the 100ms (milliseconds) range, so the clients may be able to receive updates of information in almost real time.

In addition, the short polling approach also wastes network bandwidth and server resources, because the server will respond to each request even when there is no update available. This means that, most of the time, a client's request will receive nothing back from the server.

In fact, one client making a request to the server in a 100ms interval range might create no issues, but if thousands of clients make requests in this interval range, it will eventually overload the server, so that it will simply stop accepting any of new incoming requests until the existing requests in the server's thread pool have been processed. In this case, the subsequent clients will still not be able to receive updates of information from the server in real time.

Thus, the short polling approach is not suitable for applications implementing real time transient notification architecture. Figure 12 shows illustrates workflow of short polling approach based on the flow chart schema (All flow chart diagrams in this thesis are produced by Edraw Max [193] diagram drawing tool).

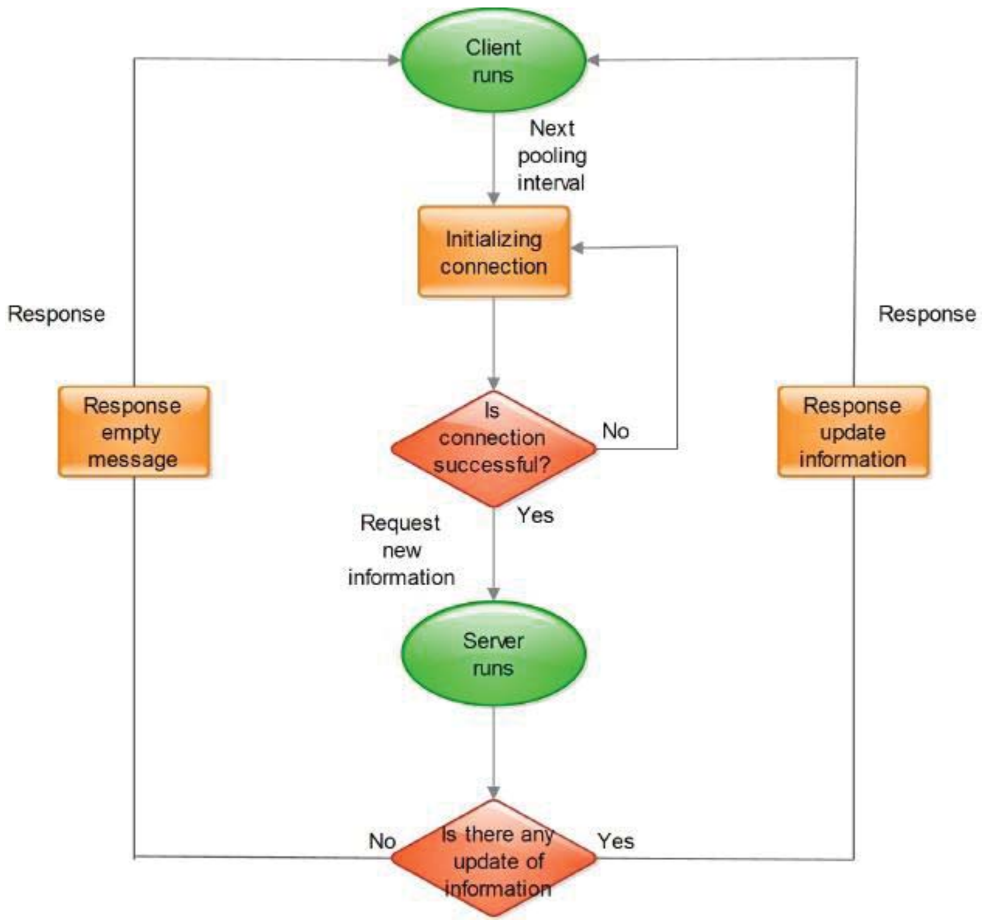


Figure 12: Workflow of the short polling approach. As the diagram shows, the connection initialization is dependent on the interval time defined in the program.

3.2.2 Long Polling Approach

The idea of the long polling approach (also known as asynchronous polling) [122] is designed to resolve problems existing in the short polling approach. In this approach, the connection and request is made by the client as with the short polling approach, but without the polling interval.

Provided the client opens the requested connection and successfully connects to the server, the server will hold each requested connection open and only respond to the client when it has an update of information, rather than responding to the client immediately. This means that every request made by the client will eventually get an update of information back from the server. Whenever the updating information is returned, the client will open up a new requested connection immediately [123].

Nevertheless, this approach has three major drawbacks. First, the server needs to consume extra resources to retain and maintain each request connection, which will increase the overhead of the server.

Second, the long polling approach is only able to deliver updates of information to the client in close to real time, rather than in exact real time. When the client side sends a request to the server side, the request will be pending in the server side, and this request will only be responded to when there is something new on the server side. However, if the server side does not have any new information within a particular time frame (default request timeout of web browser is 300 seconds), a request timeout will occur and the client side needs to issue a new request to the server side [227]. Indeed, most of the network architecture include proxy servers whose timeouts are set less than 300 seconds [228]. Thus, if the server side has updated the information during the timeout period, the clients will still not be able to receive the updated information from the server on time. Moreover, some astronomical transient projects, such as LSST and GCN, will detect and issue a lot of new transient notifications each second. The long polling approach will not be able to handle this heavy real time task, because this approach is not designed for heavy workload real time tasks. Too many requests in a very short time scale will eventually result in the server reaching its resource limit.

Last, the additional header data will also be sent back to the client with the updated information, causing additional bandwidth use. Therefore, the long polling approach is also not a suitable approach for a real time transient notification architecture. Figure 13 illustrates flow chart diagram of long polling approach.

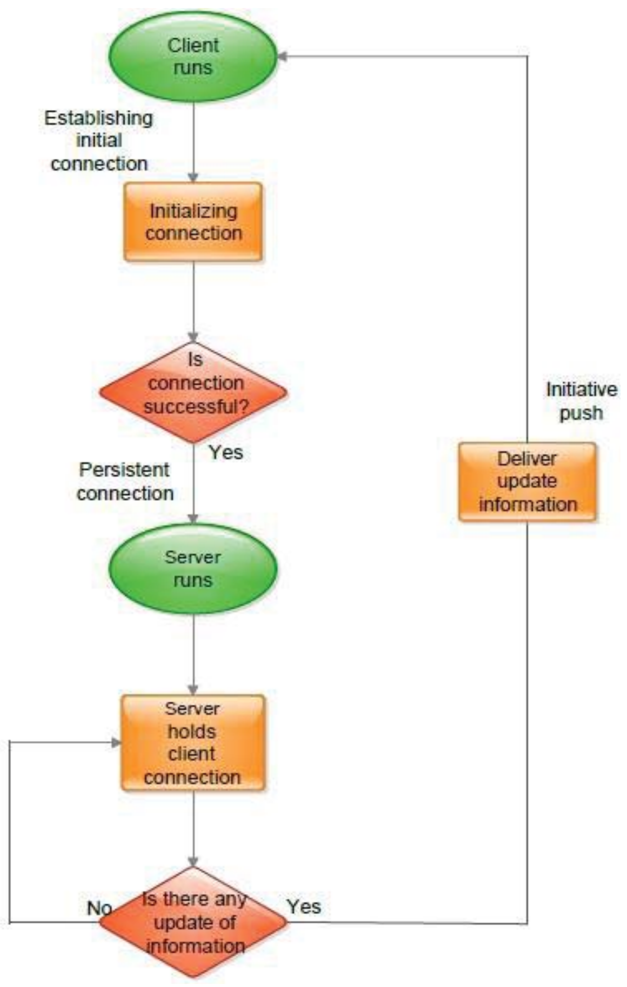

Client runs
Establishing initial connection
Initializing connection
Is connection successful?
Yes
Persistent connection
Server runs
Server holds client connection
No
Is there any update of information
Yes
Deliver update information
Initiative push

Figure 13: Workflow of long polling approach, as shown in the above diagram, the server will use an infinite loop to hold each request connection and respond to the client only when updated information is available. This flow chart diagram is produced by Edraw Max.

3.3 Push Mechanism

The push approach [124] is typically designed for those systems or architectures that need to deal with real time tasks with heavy workloads. With the push approach, the network bandwidth usage is lower than with the polling approach. This is because the connection between the clients and server will be generated only once. Whenever an update of information is available, the server will "push" the update information to the clients immediately, and the connection will stay open after the clients receive the update from the server, rather than generating a new connection to the server as with the long polling approach.

The connection between the client and the server will only close when a client sends the disconnect signal to the server, or when the server shuts the connection down for some other reason, such as a connection timeout, or if the connection is inactive for a long period of time.

The advantages of the push approach [125] are: (1) It can continuously offer high accuracy of information transport even when the server is updating information rapidly; (2) it is a convenient approach, as the latest information will be delivered to all clients automatically, so the clients don’t need to keep hunting for the newest information from the server; and (3) the push approach is much more efficient compared with the polling approach, because the client does not need to keep initiating the connection. The latest information will be delivered to clients instantly, thus, overload problems can be prevented.

In contrast, the major disadvantages of the push approach are: (1) It requires additional computation loads on the server; (2) it will incur a state space overhead, because while a large number of clients are connecting to the server, the server needs to consume lots of resources in order to maintain all of the connections; and (3) The client side will automatically receive new information from the server side without any data request operating, which means that the main control flow moves from the client side to the server side. Thus, this raises a security concern, because the server could send large amounts of irrelevant information to the client side, and the client side cannot reject receipt of the irrelevant information [229]. In other words, the quality of the received information fully depends on the server providers [230] [231]; extra development effort is needed on the client side in order to prevent irrelevant information from being received.

The first two disadvantages can be resolved by expanding the physical hardware of the single server, or by setting up a server cluster. The last disadvantage can be resolved by implementing the filtering feature on the client side. The flow chart of push approach is illustrates in Figure 14.

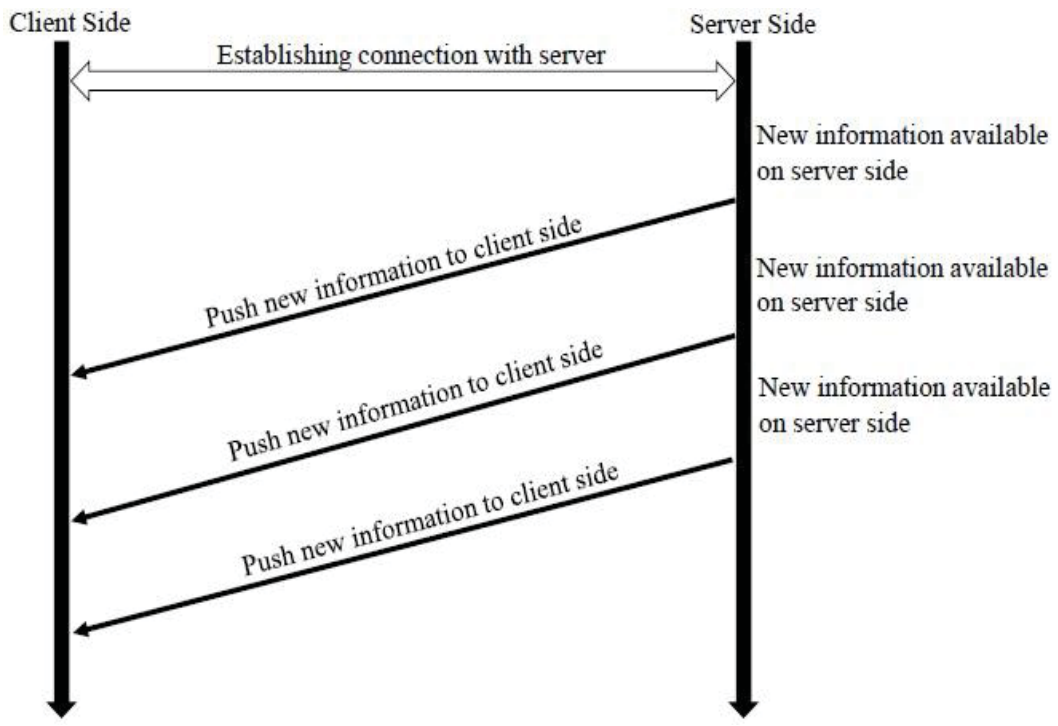

Client Side
Server Side
Establishing connection with server
New information available on server side
Push new information to client side
New information available on server side
Push new information to client side
New information available on server side
Push new information to client side

Figure 14: The flow chart diagram of push is very similar to that of long polling. The main differences are that the connection between client and server is a persistent connection, rather than a temporary connection and the server will initiative pushing updates of information to the client instead of using a request/response scheme. This flow chart diagram was producesd using Edraw Max.

3.4 Publisher and Subscriber Mechanism

The Publish-Subscribe mechanism (also referred as Pub/Sub) is a subset of the push approach. It is a computer science concept and messaging pattern typically designed for providing real time notifications to multiple destinations. In the Pub/Sub mechanism, the information that needs to be disseminated is given the term message; the messaging is produced by an entity called the publisher, and is finally consumed by another entity called the subscriber.

The Pub/Sub mechanism is based on a many-to-many communication relationship, which means that more than one publisher is allowed that can publish messages at the same time; also, each message can be consumed by multiple subscribers.

In general, any architecture based on the Pub/Sub scheme consists of three major components:

1. Publisher (also called the producer); this component creates the source of the message that is used for dissemination;

2. Subscriber (also called the consumer); this component has the ability to express their interest and consume the message from the publisher. Normally, this component is a client side application; and

3. The most important component is the message oriented middleware (also called information agents or message brokers); this component works as a middleman or a bridge that connects the publishers and subscribers together. It responds to the publishers and forwards the messages to all interested subscribers. The message oriented middleware contains lists of topics (also called channels or nodes). Each topic holds messages that are associated with it. The subscribers can make subscriptions to one or more topics of interest to them. After the subscribers have made subscriptions to particular topics, they will receive notification from the message oriented middleware whenever a new message is available on the subscription topic. This operation is performed by the event notification service in the message oriented middleware. Note that message oriented middleware guarantees that a new message will be distributed to all topic subscribers after the new message is available in the topic. However, the message notification will not be delivered to the topic subscribers immediately, because there is network latency involved, the time consumed in delivery of the message notification to the subscriber side is dependent on network traffic and network connection speed in the subscriber side. In other words, the time delay of delivery message notification varies with different network environments [232].

4. The subscribers can make subscriptions to one or more topics of interest to them. After the subscribers have made subscriptions to particular topics, they will

immediately receive notification from the message oriented middleware whenever a new message is available on the subscription topic. This operation is managed by the event notification service in the message oriented middleware.

The event notification service will manage topic subscriptions and delivery of messages to the subscribers in an efficient manner. In addition, the subscribers are also able to cancel their existing subscriptions; after the subscription has been cancelled the subscriber will no longer receive future notifications from the un-subscribed topics.

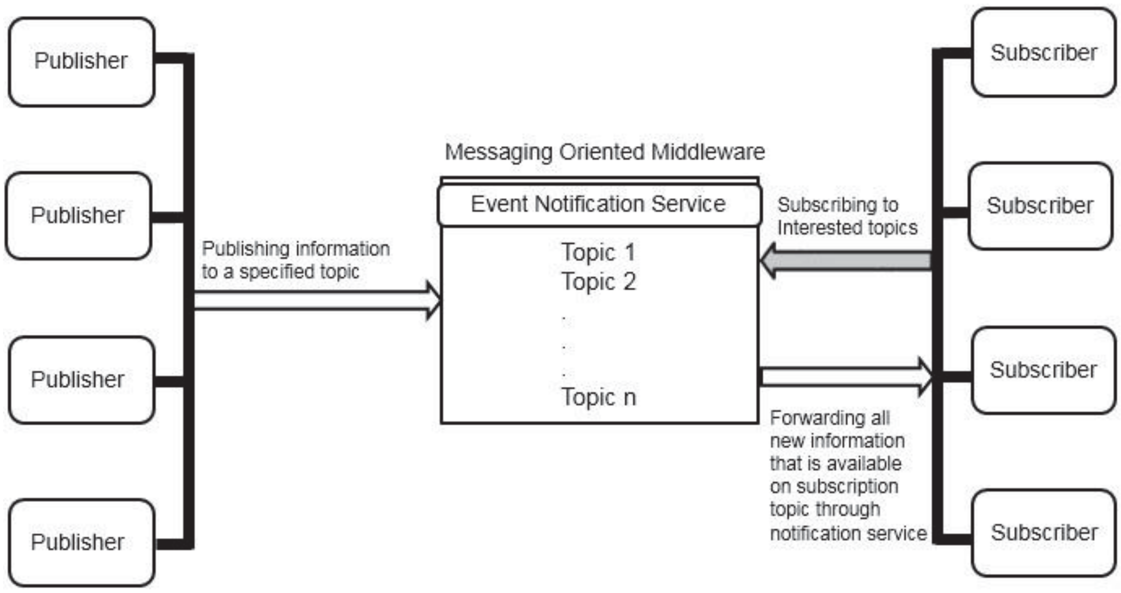

Publisher
Publisher
Publisher
Publisher
Messaging Oriented Middleware
Event Notification Service
Topic 1
Topic 2
Topic n
Publishing information to a specified topic
Subscribing to Interested topics
Forwarding all new information that is available on subscription topic through notification service
Subscriber
Subscriber
Subscriber
Subscriber

Figure 15: The above diagram shows the work flow of the Pub/Sub mode.

Another important characteristic of the Pub/Sub mechanism is the de-coupling of the publishers and the subscribers. This means that the publishers and subscribers do not need to be actively participating in the interaction at the same time, and they remain unknown to each other. In other words, subscribers do not need to keep track of all publishers and publishers do not care who will consume a message that is published to a topic. All of the interaction between the publishers and subscribers is managing by the message oriented middleware.

The advantages [126] of implementing a Pub/Sub based architecture are obvious, and include:

1. Each part within the architecture has to work independently. In other words, if a single point of failure occurs, the other components within the architecture can still work properly without being impacted; and.

2. The loose coupling nature provides scalability in the architecture, such that expanding the architecture becomes an easier task. This is because the application clients (either publishers or subscribers) can be dynamically bound to, or removed from, the architecture while other existing application clients are still running; and

3. High flexibility; even though the publishers and subscribers are implementing their connections using different programming languages, run on different operating systems, or developed by different organizations, they are still able to communicate with each other through the message oriented middleware.

On the other hand, some disadvantages of the Pub/Sub based architecture come as a side effect of the loose coupling, and include:

1. The publisher has no knowledge of the status of the subscribers and vice versa, so there is no way to ensure that everything is working perfectly on the other end;

2. The publisher could publish malicious messages, or large amounts of rubbish messages, to the topic; this can lead to all topic subscribers accessing messages that they should not be receiving and processing. In the worst case scenario, the machine running the subscriber application can be damaged and break down. This is because, in the Pub/Sub context, the subscriber will receive and process all messages delivered from the topic that it subscribes to, so if a malicious publisher keeps publishing large amounts of rubbish messages to the topic, the network traffic of the subscriber side can be blocked, and all of the system resources of the

machine running the subscriber application also eventually consumed. This is a very similar situation to a DDos (Distributed Denial of service) attack;

3. If the number of subscribers and publishers, and the quantity of information being disseminated increases, the whole architecture may become instable; therefore, some balancing strategies needs to be applied in order to handle large workload situations in order to keeps the architecture stable; and

4. The Pub/Sub mechanism requires a middleware for message management and dissemination; this extra component will increase the complexity of the architecture development.

However, the Pub/Sub mechanism is still very useful for those systems that require high information synchronisation, or require information notification within a very short timeframe; thus, a Pub/Sub based architecture is the best choice for real time transient notification architecture.

3.4.1 XMPP (Extensible Messaging and Persistence Protocol)

The XMPP (Extensible Messaging and Persistence Protocol) [127] is a popular communication protocol for message oriented middleware. It was called Jabber before and was developed by the Jabber open source community in 1999 for instant messaging, information presence and to increase the RTC (real time collaboration) in the distributed environment.

The XMPP is an XML based protocol [128]. All operations (send, receive, process) are encoded in XML format. At the initial stage, XMPP only supported a P2P (Point to Point) mechanism, but the development community has since added the Pub/Sub mechanism as an extension in the XMPP standard. The details of how the Pub/Sub mechanism works are fully described in the extension documentation – XEP0060 [129]. Figure 15 below show the messaging format of the Pub/Sub mode.

```
<stream:stream from='example' id='pubsub.someid' xmlns='jabber:client'
xmlns:stream='http_etherx_jabber_org/streams' version='1.0'>
  <pubsub xmlns=’http://jabber.org/protocol/pubsub’>
    <publish node=’examplenode’>
      <item>
        <body>Hello World</body>
      </item>
    </publish node>
  </pubsub>
</stream>
```

Figure 16: The <stream> element contains basic information of the messaging, such as where this messaging comes from, messaging id and other related information. The <publish node> element indicates which topic this messaging is sending to, and the strings between the <body> element is the actual content of the messaging.

The major advantages offered by the XMPP protocol include:

1. The XML is a well-known, platform independent, light weight and extendable exchange format. It has been used by developers in most system development for a long time. Thus, any developer familiar with XML will find it easy to move into XMPP development without much new knowledge being required by them. Moreover, any existing tools, APIs and program libraries for creating, writing, reading and validating XML data can all be used with XMPP without any, or only slight modification;

2. The XMPP based architecture is a decentralized system, which means that there is no master server, so anyone can run their own XMPP server and each individual developer or organization is able to control their own communication experience;

3. The XMPP has strong extensibility, which is a consequence of the XML format. In addition, the core stack part of XMPP only defines the basic features (such as presence, message, IQ, etc.) and the network transport logic. Extra features can be defined through the extension (i.e., Pub/Sub is the extra features of XMPP), so developers are able to create their own customized extension in order to fulfill system requirements;

4. The core XML streaming of XMPP has been formalized by the IETF (Internet Engineering Task Force) as an instant messaging and presence technology. Any of the instant messaging applications that are developed following the XMPP standard can be connected with any XMPP server; and

5. There are many XMPP related implementations (server, client, APIs) that exist for different platforms. This reduces the cost and development complexity when developing a new XMPP based architecture.

The SkyAlert and the VOEventNet projects are two typical examples that use the Pub/Sub pattern to provide transient notification to all interested recipients via the XMPP protocol. Although the XMPP is quite a powerful messaging protocol, there are shortcomings in implementing this in a real time transient notification architecture.

1. XMPP only allows for dissemination of plain text messaging or a small amount of binary data over the network; therefore, if future survey projects require the delivery of other types of messaging (such as imaging, video, etc.), a lot of extra development effort will be required;

2. The XML stream will overload when the messaging content is very large or the structure is complex, because the XMPP is designed for lightweight (less than a megabyte) message transportation;

3. There is no native XMPP support on the browser side; an extra component called BOSH is required in order to support running the application on a browser;

4. VOEvent packets in existing projects only includes basic metadata of new transient events and the file size is relatively small (a few kilobytes). Future survey projects will include much more information, such as images, full follow up links, in the VOEvent packets. The file size of each VOEvent packet will be over a megabyte, Thus, XMPP will not be a suitable protocol for transporting

VOEvent packets delivered, because it is designed for lightweight messaging delivery.

5. A high percentage of XMPP inter-server traffic is presence data, and lots of data are redundantly transmitted. In addition, XMPP also has a large overhead of delivering presence data to multiple recipients [182] [183]. These redundantly transmitted data and the overhead of the presence data can lead to unstable message delivery.

In considering the above issues regarding XMPP, it can be seen that XMPP might be adequate for current time critical astrophysics projects, but it exhibits little ability for up-scaling in order to satisfy the needs of future survey projects.

3.4.2 JMS (Java Messaging Service)

One year after XMPP was launched, Sun Microsystems proposed a messaging specification called Java Messaging Service (JMS) [130]. This specification identifies the rules of sending, receiving, managing and delivering real time messaging between two or more clients in a loosely coupled manner, and supporting both P2P (Point to Point) and Publisher/Subscriber messaging patterns by default.

The purpose of JMS is to solve the three major communication distribution issues, which are as follows:

1. Messaging reliability: The messaging delivery mechanism defined in the JMS specification ensures that each messaging is only delivered once to each messaging receiver;

2. Asynchronous communication: One application can notify another application without waiting for a response; and

3. High throughput and concurrent messaging: one important objective of JMS is to deal with high throughputs. The JMS based architecture can easily transport tens of thousands of messages per second due to its highly asynchronous nature.

Sun has also developed an API that fully implements this JMS specification. The API comes with a JAVA platform under the `javax.jms` namespace. Any developer can use this API to manipulate (create, send and receive) messaging in a distributed environment.

Unlike XMPP, which stores a message as an XML document, the JMS stores a message in a messaging object byte pattern. This messaging object consists of two parts; the message header, and the message body. The message header specifies all the meta-data related to the message, such as the destination, the delivery mechanism (P2P or Pub/Sub), the messaging ID, and so forth. The message body holds the actual message that needs to be sent.

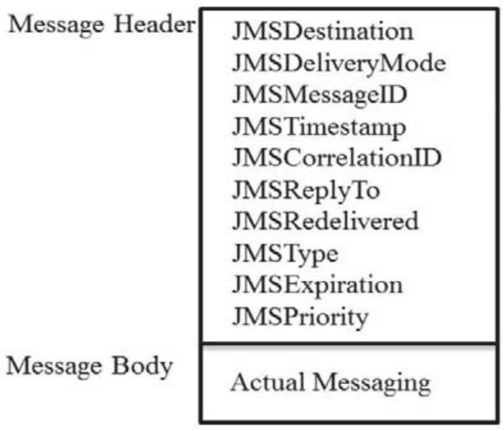


Figure 17: Messaging object structure of the JMS.

The JMS supports a large number of messaging formats; almost all types of data are able to be transported over the network through the use of a JMS API. The supported formats include stream messaging, byte messaging, map messaging, text messaging and object messaging [130].

The Stream messaging format contains a stream of primitive values in the Java programming language; the fill (write) and read operations of this type of messaging are sequential.

Byte messaging allows the exchange of binary stream (un-interpreted bytes) messaging over the network. For example, an image can be converted into a binary stream and delivered across the network.

If the system needs to exchange a key/value pair message over the network, the map messaging format can be used. In this format, the message body contains a set of key/value pairs, in which the key must be a string type and the value must be a primitive data type. The order of the entries is undefined, but the entries can be accessed sequentially by the enumerator, or randomly by name.

Text messaging can deal with any message in a plain text format, such as an XML document, a JSON document and other strings.

The object message format supports any serialised object being sent across the network. For example, sending a class object (the object must be serialized) is possible by using this format. Any information held in the class object will not get lost when it is de-serialized by the receiver applications.

Rich messaging formats are not the only strength of JMS. The JMS API also provides lots of advanced features that can used to implement a soundly performing, stable, scalable and reliable message oriented middleware. Every message oriented middleware that is implemented using the JMS API will have the same basic set of message management features (message persistence, durable subscription on topics, message cache, and so forth).

Despite the JMS providing dozens of powerful messaging features, none of the existing technologies are perfect. The main disadvantage of JMS comes from the side effect of its asynchronous nature. With asynchronous communication, the publisher application does not need to wait for a response from the subscriber application, but will instead return immediately after sending out the messaging. Indeed, most modern JMS servers (such as ActiveMQ, SonicMQ [233] and Narada broker system [234], among others) provide an acknowledgement feature in order to guarantee messaging delivery. The acknowledgement feature is a subscriber side concept; when a subscriber receives a new message from the JMS server, it will send a acknowledgement message back to the JMS server. After the acknowledgement message reaches the JMS server, the JMS server will release any resources it is holding on behalf of the said message. This feature can guarantee each message will be delivered once, and only once, to each subscriber of a topic.

Although a few disadvantages exist in JMS, it is still the most suitable technique to use for developing long term real time notification systems. Thus, JMS, rather than XMPP, is a technique that should be considered to use in transient notification architecture for transient dissemination.

3.4.3 Relative Works of XMPP and JMS

The XMPP and JMS techniques have existed for a very long time. Therefore, many studies have been carried out analysing the features, performance and usage scenarios of this two messaging techniques. Pohja [235] has evaluated how traditional HTTP based push technology can be replaced by XMPP. In his paper, he described the difference between HTTP and XMPP, and pointed out that XMPP is a much more suitable technique to use for push data than traditional HTTP based push. In addition, he also developed a score service system as a case to prove his concept; this system is a web based system, and data exchange between web browser and server is based on XMPP protocol. However, while this paper provides details of the comparison between HTTP based push and XMPP, it did not consider data transfer latency and does not present a performance evaluation of XMPP.

Xuefu and Ming [236] implemented a Web instant messaging (IM) system based on XMPP protocol; they provided a detailed description of the features and advantages of XMPP, and mentioned that XMPP has the ability to integrate with other applications and services. Unfortunately, they also did not provide any performance discussion about XMPP in their paper.

Laine and Säilä [237] described how to utilise XMPP to overcome the problems of real time communication on web systems. They carried out a study to evaluate the performance between three different techniques (HTML5, Web Socket and XMPP) in other to examine which technique is best suited for data delivery on the web. Their performance experiment was conducted in a Local Area Network (LAN) environment and three different message payload sizes (10 bytes, 100 bytes and 1000 bytes) were chosen as the experimental samples. According to their experimental results, they found that XMPP could offer better data transfer performance than the other two techniques. However, they also pointed out that there is an unexpected technical problem, which occurs when they attempt to use a message payload size of 10,000 bytes as the experimental sample, but they did not discuss the details of the technical problem in their paper.

On other hand, JMS is also a popular communication technique, and a lot of research has been carried out on this technique, as on XMPP. Farrell [238] introduced full details of the JMS technique, including its features, the objectives of JMS, its advantages and so forth. However, his paper is more like a JMS tutorial, and does not contain any implementation examples, or performance discussion regarding JMS.

Jun and Min [239] described JMS as a reliable messaging delivery service that could be extensively used to implement high performance messaging systems. They also implemented a JMS based mobile value-added business system for demonstration and concluded that this JMS based system could process 600 messagings per second. Even though they mentioned the performance of JMS, the quantitative performance analysis data and experimental conditions are not presented in the paper.

Chen and Greenfield [240] conducted a comprehensive performance study on JMS, including measuring maximum messaging throughput per second, message delivery latency and time consumed for recovery of persistent messages after server crash. The results show that the maximum messaging throughput of JMS is more than 800 messages per second and that the latency of each messaging is less than a second.

XMPP and JMS are very similar techniques and lots of existing instant messaging notification systems are build based on these two techniques. However, Foster [241] mentioned that XMPP is not suitable for use in those systems that require low messaging latency and high messaging throughputs; because XMPP is an XML based protocol, using XML as the messaging carry format will generate higher network and data processing overheads than binary protocols (note that JMS is based on binary protocol [242]). Thus, XMPP might not be ideally suited for use by LSST, as their transient data notification protocols are due to limited the scope of the messaging features, extra overheads and other XMPP issues, as discussed in Section 3.4.1.

3.5 Message Oriented Middleware

In Section 3.4, we introduced two concepts of messaging techniques - XMPP and JMS. They are, however, only a protocol and a specification, which means that they need to be implemented before being used. The message oriented middleware (MOM) is the server side implementation of these two messaging techniques. There are many JMS type and XMPP type message oriented middleware implementations available on the market, with some being commercial and some being open source projects.

This section will explore two popular open source message oriented middleware applications. These two message oriented middleware applications use different techniques for message delivery; XMPP, or JMS. Both of them are widely used as major messaging servers by a large number of instant messaging projects.

3.5.1 Openfire

Openfire [131] is an XMPP type message oriented middleware maintained by Ignites Realtime open source community and licensed under the Apache License 2.0. It is platform independent, based on pure JAVA programming language and implements most XMPP standards. It provides lots of enterprise level features such as customisability, secure sockets, database storage (mainly used for storing messages and user details) and a plugin interface, with all of these features able to be simply managed through Openfire's web based administration panel. The interoperability of Openfire is remarkable; any instant messaging applications based on the XMPP protocol are able to communicate with Openfire. In addition, Openfire is one of the few open source XMPP type message oriented middleware that supports the Pub/Sub extension (the Pub/Sub extension is not necessary in an instant messaging scheme, thus, not every XMPP type message oriented middleware supports this extension).

The Openfire server comes with various useful plugins for function extension and can run on both Windows & Linux/Unix type operating systems. Developers just need to download the corresponding operating system installation file from the official website and follow the installation guides to allow Openfire to run on the target operating system. Moreover, expanding the functions of Openfire is easy, because Openfire uses a plugin mechanism [132] to facilitate its service's expansion. When the server starts, Openfire will read all the files under the plugin director in order to determine whether there are any new plugins. If a new plugin has been detected, Openfire will access all classes that this plugin is included in and load the plugin automatically. With this plugin mechanism, developers can expand the Openfire functions in a simple way.

After the server startup, developers can develop Java based XMPP client applications using the Smack API. This API comes with the Openfire installation package and is a Java library released by the Openfire community that allows developers to fully access the features and services of Openfire.

SkyAlert and VOEventNet use this message oriented middleware to issue VOEvent packets to all interested recipients through the Pub/Sub service. Recipients can download the client side application from project websites, or write their own XMPP client application and subscriptions to corresponding topics (event streams of surveys) on the server in order to receive VOEvent packets.

3.5.2 ActiveMQ

ActiveMQ [133] is an open source application, licenced with Apache, that is Java Messaging Service compliant. It is an enterprise level message oriented middleware produced by the Apache Software Foundation that fully implements the JMS specifications and offers support for a wide range of functionalities.

There are two goals of ActiveMQ [134]. The first goal is to provide standard-based, high availability, performance, scalability, reliability and security for enterprise messaging. The second goal is to allow message-oriented application integration across as many languages and platforms as possible.

As with Openfire, ActiveMQ is platform independent and implemented by the Java language. It also provides client APIs for many other languages, covering almost all modern programming languages. This rich client APIs support feature allows ActiveMQ to be used outside the Java world, so that developers can write applications using the supported program languages to access the full features provided by ActiveMQ.

The default protocol of ActiveMQ is called OpenWire. OpenWire is a binary protocol that provides native access to ActiveMQ from different languages' applications and platforms. ActiveMQ also supports various popular protocols for connectivity other than just OpenWire.

Protocols
OpenWire
MQTT (Message Queue Telemetry Transport)
AMQP (Advanced Message Queuing Protocol)
REST (Representational state transfer)
RSS (Rich Site Summary)
Atom
Stomp (The Simple Text Oriented Messaging Protocol)
WSIF (Web Services Invocation Framework)
WS (Web Services) Notification
XMPP (Extensible Messaging and Presence Protocol)

Table 4: Full protocols of the ActiveMQ [133].

There is one remarkable protocol supported by ActiveMQ called STOMP (Simple Text Oriented Messaging Protocol) [135]. STOMP is similar to HTTP, but is typically designed to work with message oriented middleware. It provides an interoperable wire format that allows every STOMP client to communicate with any message oriented middleware supporting the protocol. Like HTTP, STOMP is a text based protocol that uses commands to communicate with message oriented middleware over the TCP/IP sockets protocol, and is language agnostic [136]. The term language agnostic refers to a message oriented middleware developed from one language or platform that is able to associate with a client application implemented in another language. The STOMP official website provides a wide range of client libraries that can be implemented by different programming languages, including Objective-C, Haxe and Javascript, among others. Developers can utilise these client libraries to access ActiveMQ through the STOMP protocol, in the case where their familiar programming language is not natively supported by ActiveMQ.

Since there are several powerful messaging features offer by the JMS, thus, it is worthwhile to consider for use JMS as a alternative technique for transient notification.

3.6 Conclusions

A computational system is necessary for time critical astronomical research, because the information of new transients needs to be disseminated to worldwide interested astronomers in real time for follow up observation and further study.

The Pub/Sub mechanism is the best choice for transient information dissemination, because Pub/Sub is an instant messaging mechanism that allows for single pieces of information to have multiple receivers. XMPP and JMS are two popular techniques that support the Pub/Sub mechanism. These two techniques are used in completely different ways to hold messages. XMPP forms messaging in an XML document, while JMS forms the messaging as a messaging object.

Although XMPP supports the Pub/Sub mechanism there are still a lot of limitations to XMPP (refer to Section 3.4.1), with limits messaging features and data transmit overheads. Thus, it may not a suitable technique to use for transient dissemination by future large survey projects due to its limited scalability.

On the other hand, the JMS offers various useful messaging features that make it possible to upscale for future survey projects. It is worthwhile to consider utilise the JMS technique for issue transient notifications. In the next chapter, I will discuss how to use JMS for transient event dissemination and provides full design of this JMS based transient notification architecture.

Chapter 4 JMS Based Real Time Transient Notification Architecture Design and Implementation

Message oriented middleware is a software application that allows real time communication in a distributed systems environment. Its role is that of a server that receives messages from one or more client applications, and immediately forwards the messages to one or more particular receiver applications. It is very useful for notification of astronomical transients and it is at the heart of the real time transient notification architecture. Moreover, the client applications can be implemented in different programming languages or hosted on different platforms, with these applications still able to communicate with each other as long as they can connect to the same message oriented middleware.

As discussed in Chapter 3, the JMS based message oriented middleware is an alternative solution that might be used instead of the XMPP based message oriented middleware, because the various messaging features offered by the JMS are able to be adapted and used by the future large survey transient notification architectures.

In this study, we chose ActiveMQ as our message oriented middleware middleware in the design of a JMS based transient notification architecture. We will discuss how to use JMS API to send and receive messages in a distributed system environment, how to deal with the transient notification, and the performance of this transient notification architecture.

This architecture design concept is not limited to using ActiveMQ as message oriented middleware middleware. ActiveMQ itself is only an application server product for message routing, and it can replaced by any JMS based message oriented middleware

middleware, such as OpenJMS [140], HornetQ [243], SonicMQ [141], and so forth. In other words, the architecture design concept proposed in this chapter is able to be applied in implementing any JMS based transient notification architecture. However, we highly recommend ActiveMQ as the message oriented middleware within transient notification architecture, because it is well implemented, and provides friendly management control UI and powerful open source JMS type message oriented middleware middleware in comparison with other open source JMS type message oriented middleware middleware.

4.1 Related Work

As Internet technology has rapidly grown, instant messaging notification architecture has become very popular; this is because this type of architecture has the ability to distribute any updated information to recipients in nearly real time [244]. In fact, instant messaging notification architecture is not only useful in the astronomical domain, and many previous studies had been carried out to investigate and develop this type of architecture in different domains. For example: Nelson [245] proposed a real time notification system that allows travellers to receive updated information about airline schedules; when a change has been made in an airline’s schedule, all system users will receive notification about this change in an automated and real time manner. However, this system is a very old fasioned system since it was proposed in 1999. Therefore, this system utilises very traditional notification approaches (website, email, paging), and does not involve any push techniques, as mentioned in this thesis.

Indeed, instant messaging was originally designed for distributed communication, but the idea of instant messaging is very useful, because it can offer nearly real time data exchange in distributed environments. Nawi et al. [246] pointed out that a traditional instant messaging application is only useful for users who are intending to contact with other users, but the instant messaging feature can be utilised for other purposes than just for chat communications. Thus, they proposed a context-aware instant messaging system with an integrated scheduling planner that allows user scheduling of appointments with other users in the contact list from the instant messaging application. In this context-aware instant messaging system, when a user has initiated an appointment, the appointee will receive a notification message with the appointment details. Afterwards, the appointee can select whether or not he/she accepts the appointment in the IM application. The user who initiated the appointment will then receive a notification message with the appointment result once the appointee has accepted or rejected the appointment. According to their findings, the employment of a well-designed context aware instant messaging system can increase collaboration between users within the same organisation, because users do not need to access their email frequently to check whether they have received any new appointments; once a new appointment is available, they will be notified at that time. However, they did not mention what protocols are utilised for the exchange data between the applications, and the performance analysis data are also not presented in the paper.

Similarly, Chang et al. [247] utilised JMS to build an instant messaging system in client/server architecture for distance education. This system allows students to exchange information and attend the classes in a virtual classroom. In this system, many virtual classrooms can be created, with each virtual classroom formed as a topic in the JMS server. The authors also developed a set of custom applications for their system; students must use this set of custom applications to access the virtual class and exchange information with other students. When a student sends a new message using the custom application, all other students who has joined the same virtual classroom (subscribed to the same topic) will receive this message in nearly real time.

Basically, instant messaging notification architecture can be applied to any system where users (human or machine) want to be notified when new information is available. Aziz et al. [248] proposed an XMPP based network management system for monitoring the status of all computers within the same internal network. There are three major components involved in this network management system; client side application, administrator application, and the XMPP server. Each computer installs a client application, and the status of all computers in the same network will be listed in the administrator application. All client applications and the administrator application are connecting to same XMPP server. When status of a computer has been changed, the client side application running on that computer will send a notification message to notify the administrator application about this change via the XMPP protocol. Subsequently, the status of the corresponding computer will be updated in the administrator application. Meanwhile, the administrator application also sends a message to the specified client application for query information regarding the computer (such as free memory space, free disk space, etc.); when the client application receives the query message from the administrator tool, it will interpret the message as a command and pattern matched with a string list that is predefined in the application. If the contents of the message match with the string list, the client application will return the required information to the administrator application immediately, otherwise, the message will be ignored and the client application will not perform any further activities. All messaging exchange between the client application and the administrator application in this network management system are uses the XMPP protocol.

Many other research efforts had been conducted on designing and developing instant messaging notification architecture systems [249] [250] [251] [252] [253]. Some are based on web environments and some are based on client/server architecture, but all instant messaging notification architectures have common feature, which is notifying the system users when new information is available. However, this study in this thesis differs from previous efforts in the following aspects:

Numerous studies have been conducted for evaluating the performance of JMS, and the size of the message payload used in those performance evaluation studies ranges from several bytes to several kilobytes (KB). However, the message payload in the time critical astronomical domain could be large; the size of a VOEvent packet generated by future projects (i.e: LSST) might more than 1 MB. Thus, determining whether JMS is capable of delivering large sized message packets over the network still needs to be researched.

In previous studies, the data structures of data used by the notification systems are not as complex as a VOEvent packet. This study will provide information on whether data with complex data structures could fit into and be carried by a JMS message object.

The event rate of time critical astronomical projects could be much higher than projects in other domains. Thereby, this study will also aim to determine whether JMS is capable to deal with high event rate situations.

None of the existing academic literatures discusses utilizing the STOMP protocol to deliver JMS messaging or benchmarking performance between JMS and STOMP. In fact, the STOMP protocol is a very useful protocol, because it is language agnostic and enables the JMS server to be accessed and communicated with by other programming languages (by default, the JMS server is only able to be accessed through using JAVA). In this study, the approach of accessing the JMS server via the STOMP protocol will be described in detail, and the performance benchmark of using the STOMP protocol to delivery JMS messaging will also be presented.

In addition, although there have been many research efforts conducted on the design and implementation of JMS based notification architectures, most of them only provide an architecture overview, or briefly describe how they could design the architecture. Thus, this chapter will describe the full design and implementation details about JMS based architecture, present the example codes for accessing the JMS server, and for publishing and receiving JMS notification messages, and discuss what the developer should be aware of during the development phase.

4.2 Case Study and Scenarios

MOA (Microlensing Observations in Astrophysics) scenarios are used as the case study in this chapter. As discussed in Chapter 2, MOA focuses on using the gravitational microlensing technique to discover planets in extra-solar systems. Gravitational microlensing is time critical astrophysics phenomenon, as it will appear for a very short duration, disappear rapidly, and never be repeated. If the transient is missed, astronomers have no opportunity to observe and study the same transient again. There are three time critical situations in which events need to be alerted and delivered to astronomy communities within a minute [254] as outlined below:

- The first situation is normal time scale microlensing events. In fact, the normal time scale events themselves are not time critical, they typically last from 20 to 40 days. However, in some situations, the light curve of a normal time scale event may change rapidly, and this anomaly may be caused by a possible planet. These anomalies should ideally be alerted within minutes of detection.

- The second situation is a high magnification event. These events are of great interest because of their high sensitivity to planets orbiting the lens stars [269], these needs to be alerted before they peak. High magnification events rise very rapidly and usually peaking within 24 hours following their initial detection. Therefore, these events should also be alerted within minutes of detection.

- The last situation involves short time scale events lasting less than 2 days. Most short time scale events are caused by free-floating planets [297]. Again, these events should be alerted within minutes of detection. However, real time detection of short time scale microlensing is very challenging for microlensing survey groups.

Apart from the three situations described above, other normal type of events can be delivered to follow up groups within an hour. After follow up groups receive transient notifications from MOA, they will pick some transients which are of interest to them and perform follow up observations to obtain more data on the selected transient/s.

However, MOA currently uses an email approach for their transient notification. This notification approach is adequate for the MOA event rate, but it is not a scalable notification approach and is not able to scale up for utilization by high event rate projects (i.e. LSST). In addition, an email notification approach may lead to astronomers to miss opportunities to observe new transients, because it is not a real time notification approach. Thus, MOA really needs suitable transient notification architecture for issuing their transient data to the follow up groups, so MOA is a worthwhile case study for this research topic.

4.2.1 MOA Data Flow System

This section introduces the internal details of the MOA data flow system before describing the actual implementation of real time transient notification architecture. This will provide a big picture of how MOA captures imaging data from the observatory through to generating the VOEvent packet. It should be noted that this MOA data flow system is not the subject of this research.

The MOA data flow system, that generates the necessary data for transient notification, involves four processes; capturing images, image subtraction, transient identification, and issuing notifications about new microlensing candidates or other interesting transient events to the microlensing community.

In the first step, MOA uses their powerful 1.8m ground based telescope to capture images from the Bulge during each observation night, and there are a total of 14 fields of Bulge observed. These observation images are called raw images and are archived in external HDDs immediately after capture by the telescope.

After the raw images are archived in external HDDs, the second step of image subtraction begins. This process is very complex and includes two steps. First, a reference image is required. The reference image is selected from the best viewing image, or a combination of the best images of a time series of observation images of a given field, and this reference image is fixed during the entire time series. Second, a difference imaging technique [137] is applied to generate the subtracted images through subtracting the observation's raw image from its corresponding reference image. Moreover, all subtracted images are stored on disk, because they are frequently used by the object identification process.

The transient identification process includes human interaction, with the observers looking at the brightness change of objects on subtracted images one by one to find interesting transient events. Once the observers discover an interesting object, they use a program to calculate the light curve by extracted photometric measurements from the subtracted images using PSF Fitting photometry; and look at the light curve by eye using a graphic tool [137].

There are three scenarios that can happen after the light curve of the object of interest has been plotted;

1. If the light curve shows clear characteristics of periodic stellar variability, the object will be flagged as a variable star;

2. If the light curve presents behaviour characteristic of a microlensing event candidate or other transient event, it will be flagged as interesting. The interesting event requires further analysis by looking at both the unsubtracted and subtracted images, checking for bad pixels, nearby saturated stars, and so forth; or

3. If the light curve can be clearly identified as a microlensing event or other interesting transient event, MOA will use a python program to generate a VOEvent packet that contains all the metadata of this new identified transient event, and then alert this new identified event to the entire microlensing community through email notification and creating a profile for this new identified transient event on their static website.

The pictures shown in Figure 18 are the second and third processes in the observatory during a real observation.

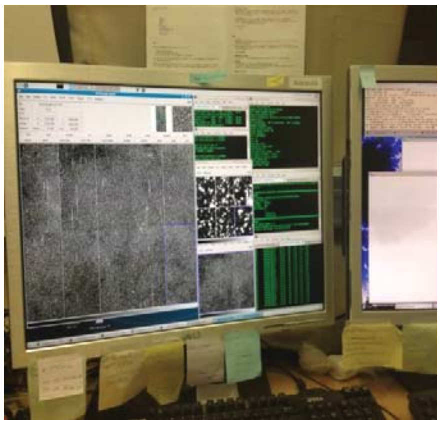
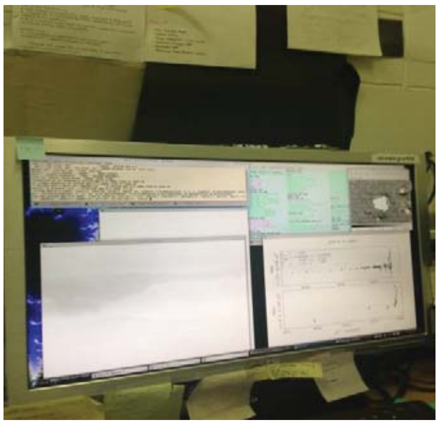



Figure 18: The left-hand picture is of the raw images captured by the telescope in a single exposure; a total of 10 raw images (the MOA camera is a mosaic of 10 CCD chips) that will be captured after a single exposure, with each raw image being 2K × 4K pixels and corresponding to one CCD chip. The right-hand picture shows the light curve generated from extracting the photometry measurement of an interesting object; the small image on the top right of the light curve is the subtracted image of the interesting object.

In addition, a single transient event may alert more than once, with the second alert also referred to as a “red alert”; a “red alert” only happens when a request of follow up observations of an ongoing transient event is received. In this case, MOA will generate a follow up VOEvent packet, issuing a second notification to the microlensing community worldwide.

In fact, the VOEvent packet is not the only data file that MOA provides to the end users; MOA also provide other data files related to newly detected transient events to end users. These data files include:

Photometry data: This data file works as an external data stream referenced within the VOEvent packet, and includes all the photometry measurements of the transient event. Each individual transient event has its own photometry measurement file. Moreover, this data file will update as more data are obtained by MOA.

Modelling data: The modeling data includes lens parameters of the microlensing event. There are two categories of lens model; the Paczynski model, and the binary model. Both of these models have common parameters, which are u_0 (impact parameter), tE

(Einstein timescale, this parameter contains physical properties of the lens) and t_0 (time of closest approach) [184] [185], but the Paczynski model also includes two more parameters besides these common parameters, which are the baseline flux value and the I magnitude. These parameters result from fitting a model microlensing profile.

On the other hand, the binary model is much more complex than the standard model, therefore, it involves more parameters; these are the values of source star radius, mass ratio, position angle, and separation. However, the binary model is not used in real time notifications, because most of the microlensing events are fitted with single lens and use the standard model. Additionally, detailed modelling of binary events is done offline.

Finder chart: MOA provides the finder chart of a transient event in both GIF (Graphics Interchange Format) and FITS (Flexible Image Transport System) format; the GIF format presents the finder chart of a transient event as a pure image. If the end users want to access the metadata of a transient event from the finder chart, they can open the FITS format using capable graphic tools, such as DS9.

Light curve plot: MOA also provides the light curve plot of transient events to end users; the light curve plot is presented in JPEG format and it might be updated when more photometry measurements are obtained.

These data files described above will not be delivered to end users via the notification system, but are accessible through the corresponding transient event profile Web Page. The MOA data flow system is illustrated in Figure 19.

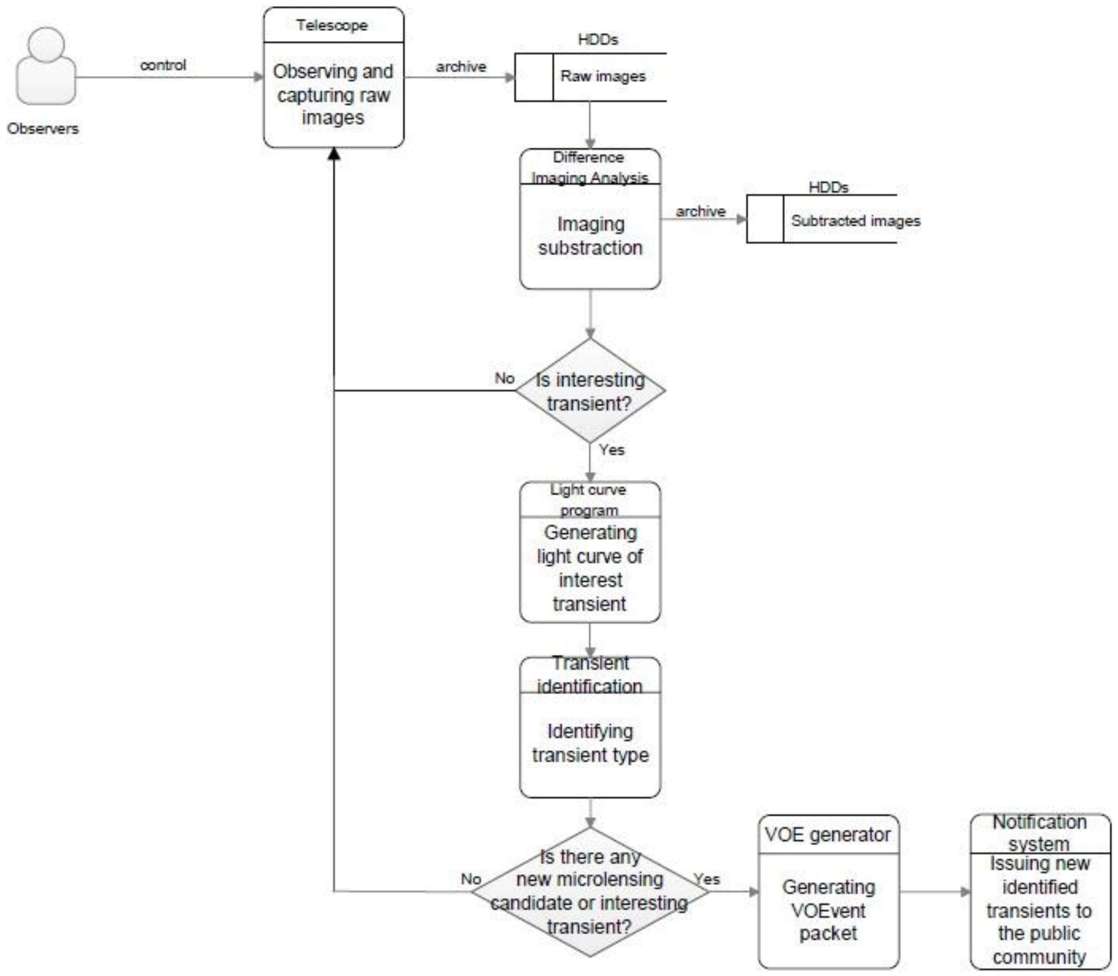

Observers
control
Telescope
Observing and capturing raw images
archive
HDDs
Raw images
Difference Imaging Analysis
Imaging substraction
archive
HDDs
Subtracted images
No
Is interesting transient?
Yes
Light curve program
Generating light curve of interest transient
Transient identification
Identifying transient type
No
Is there any new microlensing candidate or interesting transient?
Yes
VOE generator
Generating VOEvent packet
Notification system
Issuing new identified transients to the public community

Figure 19: The flow chart illustration of the MOA data flow system, this diagram was produced by Edraw Max.

4.2.2 MOA Scenarios

There are many scenarios that occur during the events' identification process. In this section, we will list some of the scenarios that commonly occur in the MOA data flow system.

1. **Unremarkable microlensing events**: Astronomers are not interested in all newly identified microlensing events, because some microlensing events are ordinary events and do not behave in a remarkable way. For example: those having no

indications of planetary microlensing. Thus, astronomers will not pay much attention to these events and the "red alert" will not be triggered.

2. **Cross referenced microlensing events**: Some of the microlensing events identified by MOA have been previously identified and issued by OGLE as well; these events are recorded in a special file called cross reference.

3. **High magnification microlensing events**: Some microlensing events could peak at high magnification (e.g., peak magnifications greater than 20). These high magnification events occur relatively rarely, but astronomers are very interested in them because high magnification events allow for the detecting of extra solar planets in an efficient manner as they involve the small use of resources over a relatively small amount of time [138]. Figure 20 shows a light curve example of a high magnification microlensing event. Moreover, a "red alert" might be issued for some high magnification microlensing events.

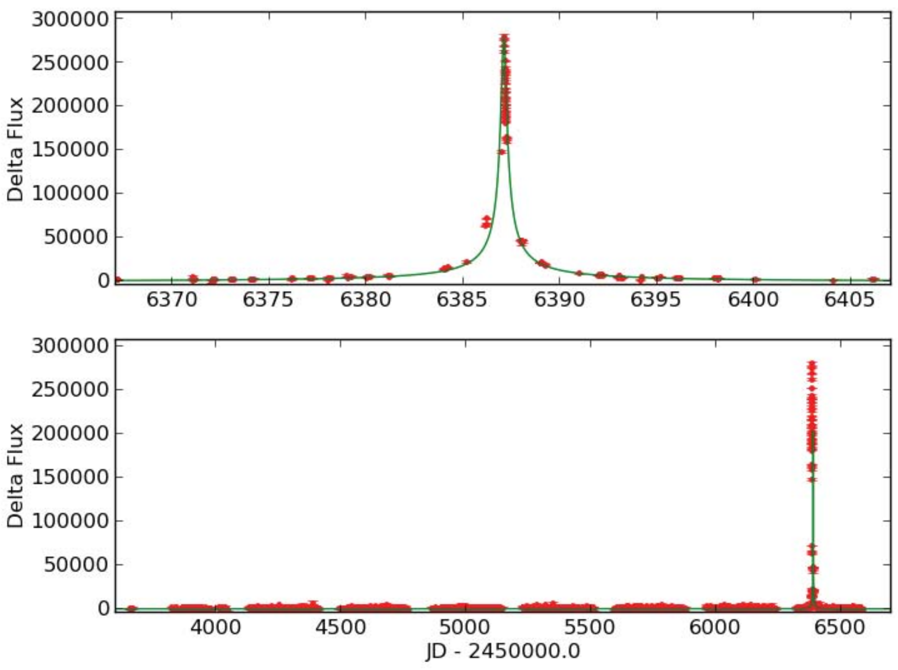
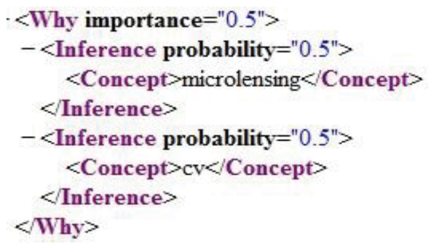


Figure 20: A typical high magnification microlensing event - MOA 2013-BLG-220.

4. **Anomaly events**: Some non-planetary companions might produce a light curve that looks like a microlensing light curve. For example, there are two types of binary lens: star + planet and star + star. Sometimes a star + star binary lens event can produce perturbations that mimic planetary signals, which will cause an ambiguity between the planetary and binary lens interpretations. These events are also referred to as anomaly events and require more study before being issued to the microlensing community.

5. **Initially unclear transient events**: For some transient events, it may not be clear as to whether they are a CV (cataclysmic variable) or a microlensing event at the time of detection. In this case, MOA will assign them the probability of transient characteristic in the VOEvent packet and still issue an initial transient notification to the microlensing community. When the characteristic of the transient have been confirmed, MOA will update the VOEvent packet, adding a description in the new VOEvent packet saying that this event is now believed to be microlensing, and issue this updated VOEvent packet to the microlensing community. In the current stage, all event classification processes are done manually; observers will look on the light curve plot of candidate events one by one to identify event types and assign the characteristic probability to the event. It is possible that an event may be incorrectly classified at the initial stage; in this situation, MOA will create a new VOEvent packet, which contains the correct classification values and alerts this new VOEvent packet to the microlensing communities. Note that event type classification processes, characteristic probability assignment and correction of incorrect classified events are not the subject of this thesis.



```
- <Why importance="0.5">
  - <Inference probability="0.5">
      <Concept>microlensing</Concept>
    </Inference>
  - <Inference probability="0.5">
      <Concept>cv</Concept>
    </Inference>
  </Why>
```

Figure 21: The VOEvent fragment of an unclear characteristic transient event; 0.5 means this transient event is 50% microlensing and 50% CV. After the characteristic has been confirmed, one of the <Inference> tags will be removed and the probability value of the retained <Inference> tag will be reassigned to 1.0.

6. **Misidentified transient events**: In some situations, a cataclysmic variable could be misidentified as a microlensing event and this misidentified transient event issued to the microlensing community.

7. **Microlensing events**: If the transient event can be clearly identified as a microlensing event, MOA will generate a VOEvent packet that contains all of the metadata of this microlensing event and issue it to the microlensing community immediately.

All of the above scenarios can be handled within the transient notification architecture that we implemented, the notification processes pipeline is completely automatic in our architecture, and new transient event information can also be processed by receiver applications (subscribers) automatically as well.

4.3 Full Design of Transient Notification Architecture

This transient notification architecture is primarily designed for MOA to issue their observations of new transient events to the follow up groups and the public domain. In this section, we will describe the full implementation detail regarding the MOA transient notification architecture.

4.3.1 Overview of Transient Notification Architecture

One important goal of transient notification architecture is the delivery of newly transient information to all interested end users in almost real time. However, this is not the only goal of our transient notification architecture. There are other goals that we also want to achieve through this architecture, as detailed below.

Delivering new transient information on multi-streams: The new transient information should be accessible from as many routes as possible. For example, social media networks are very popular and used by lots of people every day, so it is worthwhile posting new transient information on social media networks, such as Twitter and Facebook. This means that, whenever a new transient event is published, those end users who have “followed” the MOA account are able to retrieve the new transient information instantly via the corresponding social media applications.

Transient notification on mobile platforms: The mobile device has become a very powerful tool for retrieving information from the Internet; every smart mobile device has a “push” mechanism that allows the delivery of information to device users in almost real time. Thus, the architecture should have the ability to interact with mobile devices and deliver new transient information on mobile devices. The implementation details on a mobile platform will be discussed in Chapter 5.

Associated with external customised applications: The external end users should be able to develop and connect their own customised transient receiver applications within our transient notification architecture in a simple and convenient manner. MOA encourages users to develop their own applications instead of using one general event receiver application provided by MOA. This is because the general event receiver application will deliver all new transient information to the end users, regardless of their interest in those particular events. When transient notification frequency is very high, it is very difficult to filter out the event of interest. However, with a customised application, end users can define their own filter and transient information process logic inside the application.

Ease of secondary development and maintenance: The transient notification architecture must be able to extend its functions, aggregate new applications and be easy to maintain in the future. All of the components within the transient notification architecture should have clear relationships, which means adding, maintaining or removing a single component from the architecture must not affect other ongoing components. In addition, reusability is a very important factor to evaluate the design of an architecture, and a reusable architecture could reduce development time and efforts for implementing similar architecture. Thus, the event notification architecture that we have proposed here should be able to be easily reused for secondary development with only small modifications required. We also develop common publisher codes and common subscriber codes in order to simplify the steps of secondary development. Both the common publisher and subscriber codes are open source and can be downloaded from MOA's website. These codes can be utilised for publishing and receiving messaging to/from any JMS servers. In fact, there are only two steps required while other projects intending to implement similar event notification architecture for distributing their event data. Each step is as described below:

1. Setup the JMS server, and then create the topic(s) in the JMS server for event publication and user subscription (subscribing topic(s) for receiving event notifications) based on their project requirements. Please note that the JMS server is not limited to ActiveMQ; it can be any JMS server available on the market, such as SonicMQ, HornetQ, and so forth.

2. Download the common publisher codes and common subscriber codes, and specify the server IP, port number of the server and topic(s) name/s in the codes.

For those projects which are just required to delivery all event data to their users, it is only needed to complete the two steps as described above. However, one important thing to be aware of is that this architecture design, and both the common publisher and subscriber codes, aim to provide a fundamental framework for event data delivery, however, each project has its own project requirements. Therefore, when projects require advanced features (i.e: analysis event based on received data, driving robotic telescope for follow up observation, etc.) after receiving event data from the JMS server, additional development efforts are still required to be implemented on top of the publisher and subscriber codes. As mentioned above, both the common publisher and subscriber codes are open source, so they can be freely modified.

Figure 22 provides an overview of how this architecture works. Whenever the MOA publishes new transient information, all applications that connect with the message oriented middleware middleware will be notified, and receive the new transient information from the message oriented middleware middleware immediately.

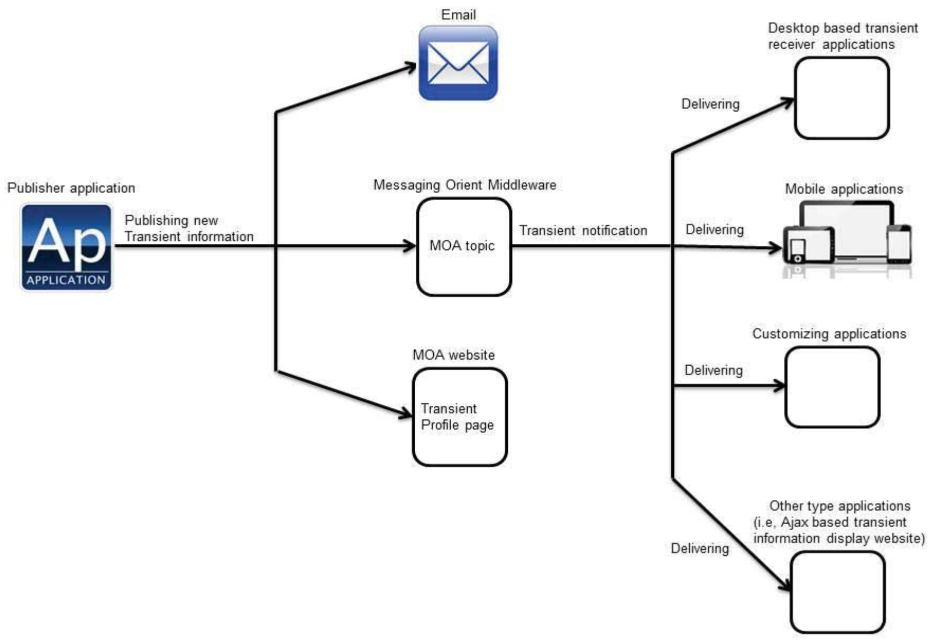

Email
Desktop based transient receiver applications
Delivering
Publisher application
Messaging Orient Middleware
Mobile applications
Ap APPLICATION
Publishing new Transient information
MOA topic
Transient notification
Delivering
Customizing applications
MOA website
Delivering
Transient Profile page
Other type applications (i.e, Ajax based transient information display website)
Delivering

Figure 22: Workflow of transient notification architecture.

From the diagram, we can see that every application within this transient notification architecture works individually. Thus, if a single receiver application has been modified, removed, or experiences a breakdown, this will not affect the rest of the applications. In fact, adding a new application into the architecture is also a simple task; the end users only need to develop their application, connect with the message oriented middleware, and everything is ready to go.

Furthermore, we still retain the old email notification and posting of new transient information on the MOA website, because some astronomers might still prefer to access new transient information using these two approaches.

4.3.2 JMS Type Message Oriented Middleware Selection

The message oriented middleware middleware is the heart of the transient notification architecture. There are many JMS type message oriented middleware middleware products available on the market, such as ActiveMQ [133], Oracle Weblogic [139], OpenJMS [140], and SonicMQ [141], among others. We need to pick the one that is most suitable for the MOA project from these various JMS type message oriented middleware middleware products. After research and product feature comparison, it was decided to use ActiveMQ in the MOA transient notification architecture based on the six reasons outlined below:

1. **Open source:** As described in Section 3.5.2, ActiveMQ is a completely open source product under Apache license. This open source product is allowed to be used for free and users can make any modifications to its original design. Thus, we can legally redistribute and improve the source codes of the product when necessary.

2. **Development flexibility:** ActiveMQ supports all mainstream programming languages. This rich programming language interface reduces difficulty of development, because end users can develop custom transient receivers based on the programming language they are already familiar with rather than having to learn a new one.

3. **Network extensibility:** As we know, many organizations have very restrictive firewall policies. In the most extreme cases, the firewall of the research community only allows access to the Internet via HTTP/HTTPS protocols. For those cases, the transient receiver applications behind the firewall are not able to connect to the remote message oriented middleware, because the message orient middleware uses very special port numbers. For example, the default protocol utilised by ActiveMQ is Openwire and it uses 61616 as the default port; this port might be blocked by some firewalls.

Fortunately, ActiveMQ supports full network protocols; if the default port has been blocked by the firewall, the receiver applications can retrieve new transient information from ActiveMQ through the HTTP/HTTPS and RESTful protocols, these three types of protocols will be permitted by any types of firewall.
This aspect makes ActiveMQ accessible under any environmental context and is one of the most important reasons why MOA is designed to use ActiveMQ as its message oriented middleware for transient notification.

4. **Rich user authentication mechanisms:** The security of message oriented middleware middleware is a very important element that also needs to be considered. ActiveMQ provides strong user authentication mechanisms, which can prevent unauthorised users sending messages. This authentication mechanism will be discussed in detail in Section 4.2.4.

5. **Handling large throughput:** ActiveMQ has the ability to handle large throughput (many receiver applications connecting to ActiveMQ at the same time) situations.

6. **Strong community support:** A lot of developers are involved in this project and are frequently improving ActiveMQ based on user feedback. In addition, most solutions for development problems are able to be found from within the ActiveMQ community.

ActiveMQ also offers lots of extra features besides those described above, and it satisfies all of the requirements needed for implementing transient notification architecture.

4.3.3 ActiveMQ Deployment

The Apache site provides two different versions of ActiveMQ distributions that correspond to the Linux/Unix and Windows operating systems. Both of them can be found on the ActiveMQ download webpage [142]. The distribution file is in the form of a ZIP file, and the user simply downloads the version that is suitable for use with the target operating system, unpacks it, and a root directory called apache-activemq-x.x.x (where x.x.x is the version number) will be retrieved after the ZIP file has been unpacked.

There are seven sub directories within the root directory; which are `bin`, `conf`, `data`, `docs`, `example`, `lib`, and `webapps`. The `bin` directory contains an executable file of the ActiveMQ server. Moreover, there is a sub directory called `macosx` under this bin directory with Linux/Unix distribution, which includes the executable file that is typically targeted for the Mac OSX operating system.

The `conf` directory includes all configuration files regarding the ActiveMQ server, such as the main configuration of the server, web console configuration and scalability configuration. These files are formatted as XML documents and can be edited through any of the text editor tools.

The default database used by the ActiveMQ is called kahadb [143]. It is a file based persistence database that uses less file descriptors, is optimised for fast messaging persistence, and offers faster recovery ability for messaging recovery. This database can be found in the `kahadb` directory within the data directory. In addition, `kahadb` uses a Btree data structure to maintain and store all of the important information related to the message broker. For example, every message that has been published to the topic in ActiveMQ will be recorded in this database.

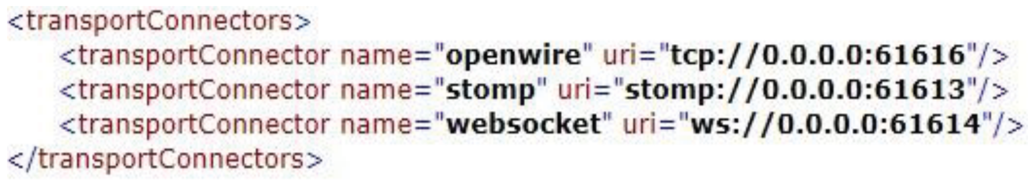
The `doc` directory only contains one html file that describes all of the features and updates of the current version of ActiveMQ. Apache also provides some source codes as a starting guide for those developers who have never used ActiveMQ before; these source codes can be found within the example folder.

The lib directory includes all the programming library files that will be used during the development period. These library files are formed in jar (Java ARchive) format and they can only be used within Java development. Examples of connecting to ActiveMQ using other programming languages will be described later in this chapter.

The last directory, `webapp`, includes all files that relate to the web management console. The web management console can be accessed through the Internet browser after the ActiveMQ has been launched.

Configuring ActiveMQ for transient notification architecture

There is one modification on the ActiveMQ configuration file that needs to be done before deployment. This is because ActiveMQ only enables the Openwire protocol within the initial configuration file, as mentioned in Section 3.5.2. There are only Java and C++ implementations available for the Openwire protocol. In other words, this protocol can only be connected through the Java or C++ program. Thus, we need to enable two extra protocols that allow the end users to develop the programs in other programming languages. Additional protocols can be enabled by adding an extra `<transportConnector>` tag under the `<transportConnectors>` tag within the configuration file.



```
<transportConnectors>
    <transportConnector name="openwire" uri="tcp://0.0.0.0:61616"/>
    <transportConnector name="stomp" uri="stomp://0.0.0.0:61613"/>
    <transportConnector name="websocket" uri="ws://0.0.0.0:61614"/>
</transportConnectors>
```

Figure 23: Adding new supported protocols in ActiveMQ configuration file.

We enable both STOMP and WebSocket protocol [186] supports in our ActiveMQ server, STOMP is a language agnostic protocol and it can be connected through all mainstream program languages, such as Python, Perl, Flash and Delphi, as we already described in Section 3.5.2. The WebSocket enables end users to implement a dynamic transient receiver website through using HTML5 websocket protocol, because the WebSocket protocol supports asynchronous messaging for web pages and it can be connected through JavaScript.

Solving the firewall issue

After new network protocols support were added, we deployed our ActiveMQ server within the Massey University system. However, we encountered our first issue after deployment, which is the firewall issue. This occurred because Massey University, like many organizations has a very restrictive firewall policy. There are limits on the inbound and outbound ports allowed to open in order to prevent network attacks and other potential issues. However, the Openwire, STOMP, and Websocket protocols use very special ports, as shown in Figure 23. Requesting to open these ports will not be approved by the Massey University system, resulting in our ActiveMQ server being inaccessible from outside. Therefore, we need to find somewhere else to host our ActiveMQ.

After some research was carried out and discussion engaged in, we realized that virtual computing will be a good solution to resolve this issue. Virtualisation is the technique that makes cloud computing possible, virtualizing means converting physical hardware resources to virtual hardware resource for the efficiency of its consumption. Therefore, virtual computing also known as on-demand computing. In virtual computing context, users do not need to spend lots of money to purchase their own computer to fill the role of a server, and they can also upgrade their virtual computers on demand, which means they can boost the virtual computer performance, storage size (HDD), memory size, processes and applications whenever they need.

The virtual computer works in a completely different way from a typical webserver hosting service. In the web server context, only limited resources of the web server are able to be used and users can only interact with the web server through an FTP (File Transport Protocol) service. For example, users are only able to use those applications that are pre-installed by the web server providers. Installing new applications on the web server is not possible.

In contrast, virtual computing provides more flexibility for the users, because they are granted root permission and have full access control of the virtual computer. Thus, users can perform any operations in the virtual computer as they would normally do in their local computer, such as install new applications, manage network access permissions, set up additional user accounts, and so forth. Moreover, the virtual computer user does not need to deal with hardware or any replacements or physical upgrades, because the virtual computing providers are responsible for all of the troubleshooting, data backup, maintenance and protection of the underlying physical server; when virtual computing providers update their physical server, the performance of the virtual computer will be automatically updated as well.

Virtual computing is a very convenient technology and it fulfills all hosting requirements of ActiveMQ server. For these reasons we decided to host our ActiveMQ server on a virtual computer on a cloud service instead of within the Massey University system.

We chose Amazon as our virtual computing provider and created a virtual computer through the Amazon EC2 (Elastic Compute Cloud) [144] [145] service. Amazon EC2 is a Web service that provides scalable computing capacity in the cloud, and it is the core part of the Amazon cloud computing platform. The EC2 system also reduces the time required to create a new virtual computer to minutes; it provides a simple web service interface that allows users to create, launch, resize, and terminate the virtual computer as needed with only a few mouse clicks. Additionally, EC2 offers very flexible charge rate options for users. For example, users can pay by hourly for activity on the virtual computer, or pay by the actual capacity used, among other options. This flexibility in charging options is ideal for the research, because it saves lots of research funding and is one of the reasons why we have used the Amazon EC2 service.

In order to use a virtual computer through the EC2 service, a virtual computer instance (also known as an Amazon EC2 instance) needs to be launched. The first step of the launch process is to boot an Amazon Machine Image (AMI). The AMI is a software configuration template that contains a set of software (operating system and applications) required to launch the instance; the user can either select an AMI from the AMI list provided by EC2 or use their own AMIs. The AMI we use in this research is Microsoft Windows Server 2008 R2 (64-bits architecture).

Next, an instance type needs to be chosen. The instance type is the hardware configuration of the EC2 instance, including the CPU, memory, HDD capacity and network performance. There are various different instance type options available, with the user needing to select one from the instance type list. In this research, we chose the basic instance type, because MOA will not generate large network throughput compared with other future surveys, and user groups are relatively small. Thus, the basic instance type is sufficient for this research.

The EC2 instance is ready for launch once the instance type has been chosen. The login process can be done through using the RDP (Remote Desktop Protocol) shortcut file that downloads from the EC2 web service interface panel.

There is one more step required after a successful login, which is setting up the inbound policy to open ports 61613, 61614 and 61616 in order to make our ActiveMQ server accessible from outside through the STOMP, websocket and OpenWire protocols.

After all of the above steps have been completed, we run our ActiveMQ server, writing a Java based connection testing program; a connection is successfully established with the ActiveMQ server.

After all of the firewall and connection issues have been solved, we are ready to enter into the second stage of the research. In the section, we will discuss the topic design of the MOA transient notification architecture.

Note that the decision to run the system on the EC2 cloud was a pragmatic one to avoid having to negotiate special port opening with the Massey system administrators. Futher projects such as LSST may well have the resources to deploy their own hardware to run their messaging system and set their own port access policies.

4.3.4 Topic Design and Protection

As discussed in Chapter 3, the message sender (publisher) is not directly communicating with the message receiver (subscriber) in the Pub/Sub paradigm. Any communications between the message sender and message receiver need to go through the topic(s) that are created in the message oriented middleware middleware. The term topic is a terminology used in the JMS Pub/Sub paradigm (also referred to as channel in XMPP terminology); many topics can be created in message oriented middleware middleware, and each topic will consume approximate 29.5KB of physical memory. A topic can be seen as a message container, which holds all messages related to it. The subscriber must subscribe to a topic before it can receive messages send by the publisher. Thus, we need to create topics in the ActiveMQ before any transient information publication. Additionally, the topic must be well designed and in order to do so we consider three aspects, as outlined below:

1. The topic created in ActiveMQ must be easy to maintain; creating new topics in the ActiveMQ should not affect the existing topic subscribers. Furthermore, the topic layout in ActiveMQ should not require big modifications in the future.

2. The JVM will allocate a limited amount of memory to each Java application (ActiveMQ is a Java application); this limit is specified during the application startup. Each topic created by ActiveMQ will consume some of the available memory (approximately 29.5KB), when the memory limit of the application is hit, an out of memory exception will be thrown. Of course, this issue can be easily solved by increasing the memory limit through specifying parameters − `Xmx` before the ActiveMQ startup. In other words, increasing the memory limit means that more physical memory will need to be allocated to the ActiveMQ. However, there are other applications we need to run in the same host machine. Thus, memory usage is one aspect we need to consider during topic design.

3. The last aspect of topic design is that the topic should continuously work for receiving present and future transient information without requiring frequent updates.

The first idea that was considered for topic design was to create a topic for each new transient; that is, a new topic will be created in ActiveMQ after the new transient has been detected by MOA, and this topic will use the external ID of the transient (e.g., 2014-BLG-001) as the topic name. There are two major drawbacks of this topic design method. First, a large number of topics will increase the difficulty of maintaining the ActiveMQ. Most of the topics will only be associated with one message and will never be used again, because most of the transients will only alert once and not have any "red alerts". Therefore, The ActiveMQ administrator needs to check and remove unused topics frequently for maintenance purpose. Indeed, tracking unused topics from a large number of topics is a very inconvenient and time consuming task.

Second, a large number of topics will also increase the difficulty of maintaining the ActiveMQ. Most of the topics will only be associated with one message and will never be used again because most of the transients will only alert once as an initial notification and not have any "red alerts" attached to them. The ActiveMQ administrator needs to check and remove unused topics frequently in order to free memory space for future transients, which is very inconvenient. Moreover, tracking unused topics from large numbers of topics will also be an inconvenient task and will waste time.

Third, the topic name is based on the external ID of transients in this method; end users must know the ID of new transients in advance before they can subscribe. However, there are only two ways that end users can obtain the transient ID.

The first way is to check new transient IDs on the MOA website, but this will mean that the new system has not resulted in any change from the previous situation; end users will still need to visit the MOA website frequently in order to check whether or not new transients are available. Therefore, there is no point in implementing this transient notification architecture. The second way is to write a program to automatically check for updates on transients from the MOA server using a predefined timeframe (polling); when a new transient is available on the server, the program will fetch the ID of the transient and add it into the subscription program. However, this will increase the workload of the MOA server significantly and generate a lot of extra lines of code in the subscription program. In addition, with this second method, the new transient information will still not be able to delivery to the subscriber in real time due to the polling mechanism involved. This topic design method also requires a lot of modifications of the subscription codes. For example, the end users need to modify their subscription to remove those topics that are already removed from the ActiveMQ, which will create the same situation as with the ActiveMQ administrator. Thus, this topic design method has been discarded.

The second idea we examined was for each end user to have their own topic created in the ActiveMQ. The new transient information will be sent to all of the user topics after detection; this method will overcome all of the issues in the previous topic design method, and works perfectly with the current user base, but it still has a potential memory usage issue; if the user base increases greatly in the future (this is an unlikely case, but we still need to consider it) the same problem as the first drawback of the previous design method will occur. The main reason for this is that we only have a single host machine with limited memory. Thus, this topic design method is not suitable for our research.

After these two topic design methods were discarded and further discussion was entered into, we finally developed a solution for the topic design; using only a single topic to deal with all new transient publications. With this method, the topic creation will only occupy a small amount of the memory of the host machine. Thus, the “out of memory” exception caused by too many topics will not occur again, even if our transient notification architecture runs for a very long time.

From the users’ point of view, this method also provides two major advantages to end users. The first advantage is flexibility, as each microlensing research group might have their own transient analysis logic after they receive transient information from MOA, so we only provide common subscription codes (an example of the subscription codes will be given and discussed in Section 4.4) for message receiving, and leave the detailed implementation of the analysis part for the end users rather than implementing a fixed transient analysis application for end users.

Another advantage is convenience; whatever update or change has been made in the server side, the same subscription codes are still able to be consistently utilised for acquiring new transient information from the same topic without any modifications. This one time development characteristic is one important benefit delivered by using a single topic for all transient information publication.

This single topic method definitely overcomes all issues that existed in the previous topic design methods. Thereby, we finally decided to use this topic design method for transient publication in our transient notification architecture.

The transient information publication topic has been named “`moa.voevent`”, mainly for indication purposes. This is because we might create new topic(s) for feeding other types of transient information to end users rather than just using VOEvent packets in the future. These other topic areas might include photometry, calibration data, and even serialised strings of finder chart images. Naming topics in this way will indicate to end users what type of information the specified topic is providing (i.e., moa.voevent provides VOEvent packets of new transients to end users), so end users can subscribe to corresponding topics that match their areas of interest and only receive the particular information they want to receive.

In addition, if other survey groups want to collaborate with MOA and create their own topics in our transient notification architecture, the topic name will also be useful for separating different surveys, and end users will know where the source of the transient information is come from.

Security Protection

One of the important issues in regards to the Pub/Sub based architecture is the possibility that anyone can publish messages to the topic. This could result in topic subscribers receiving malicious messaging from the topic. In fact, network data exchanges are required when receiving messaging from the message oriented middleware. Therefore, in the worst case scenario, a lot of network bandwidth will be consumed in the subscriber side when a large amount of malicious messages with large sized payloads are published to the topic. Thus, the main security goal of this system is the prevention of malicious messages being received by the subscribers. In other words, the subscribers should only receive transient event data from the message oriented middleware.

Indeed, the message selector feature offered by JMS could be one solution to resolve this issue; the message selector allows a subscriber to specify the messages they are interested in [255]. After the message selector feature has been enabled, the message oriented middleware will examine each message sent from the publisher in order to determine whether or not the message is of interest to the subscriber; if the message does not match the interests of the subscriber, the message oriented middleware will not delivery this message to the subscriber [134]. In other words, the work of filtering messages is assigned to the message oriented middleware, rather than the subscriber application. This means that the message filtering work is done on the message oriented middleware. Thereby, the workload of the message oriented middleware will increase significantly; this is because, if large amounts of malicious message have been sent by a malicious subscriber, the message oriented middleware will still examine all of these un-useful messages one by one and discard them. As a result, a lot of resources will be wasted in order to perform this unnecessary operation. Due to this fact, we do not choose the message selector as the security mechanism for our system protection.

After several discussions, the best solution for system protection is that the "*moa.voevent*" topic should only allow MOA staff to publish to it; other end users should only be able to subscribe to, and receive messaging from, the topic. In order to achieve this goal, a security mechanism needs to be applied in order to guarantee that only MOA staff can send messaging to the "*moa.voevent*" topic. We note that this topic protection concept must be applied on all transient notification architectures.

In this section, we will discuss how to achieve the goal mentioned above using the ActiveMQ security mechanism. Note that each different JMS message oriented middleware provides its own security mechanism for topic protection that may differ from the ActiveMQ security mechanism.

There are three types of security mechanisms for topic protection supported by ActiveMQ; simple authentication, JAAS (Java Authentication and Authorization Service) authentication, and custom authentication. After any one of these three security mechanisms has been applied, each end user needs to create a user account within the ActiveMQ security domain before proceeding with any association with the ActiveMQ (connecting, publishing or subscribing). Moreover, each user account must have a unique user name, because ActiveMQ does not allow the same user name to be used by more than one user within its security domain (this distinct user name nature is common in all message oriented middleware).

The simple authentication can be implemented very easily by adding a "`simpleAuthenticationPlugin`" into the ActiveMQ configuration file. However, with the simple authentication mechanism, all registered users are able to publish to the "`moa.voevent`" topic, because this mechanism will only check whether the login user is valid or not. After the user has successfully logged in to ActiveMQ, they can perform any operation with the "`moa.voevent`" topic. However, this simple authentication mechanism does not allow us to achieve our topic security goal.

The JAAS [256] is a much more powerful and flexible authentication mechanism in comparison with the simple authentication mechanism. In this authentication mechanism, each user will be assigned a role, and the topic is associated with the user role rather than the actual user, so each individual topic can be configured so that only users in specific roles can publish messaging to it. Figure 24 below illustrates how the JAAS authentication mechanism works.

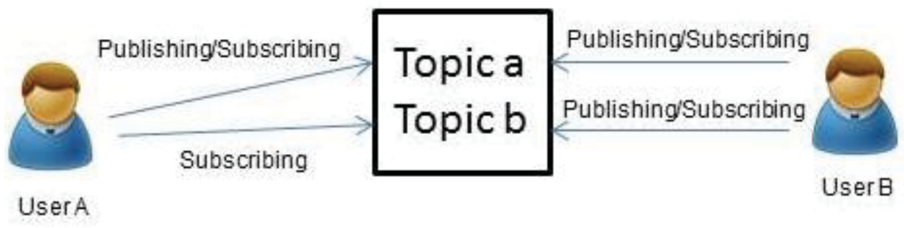


Figure 24: With JAAS, both user A and user B will be assigned a role. We can control the topics to allow both the roles of user A and user B to be able to publish messaging to topic a, but only the role of user B to publish messaging to topic b.

There are only two steps required to activate the JAAS mechanism in ActiveMQ. First, two extra files called `users.properties` and `group.properties` need to be created in the conf directory (the conf directory is located under the ActiveMQ root directory); these two files can be created by any type of text editor application and changing the file extension from `.txt` to `.properties`. The users.properties file contains user name and password information about all register users; each individual register user is represented in one line and the data format is "`username=password`". The `group.properties` file contains all of the roles information, with the group.properties file being similar to the `user.properties` file. Each role is represented in one line, but each role can be assigned to more than one user through the use of a comma to split different users; the data format of this file is "`role name=username,`username". Figure 25 shows an example of both `user.properties` and `group.properties` files.

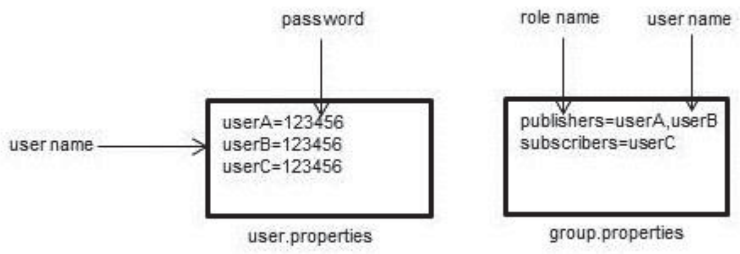
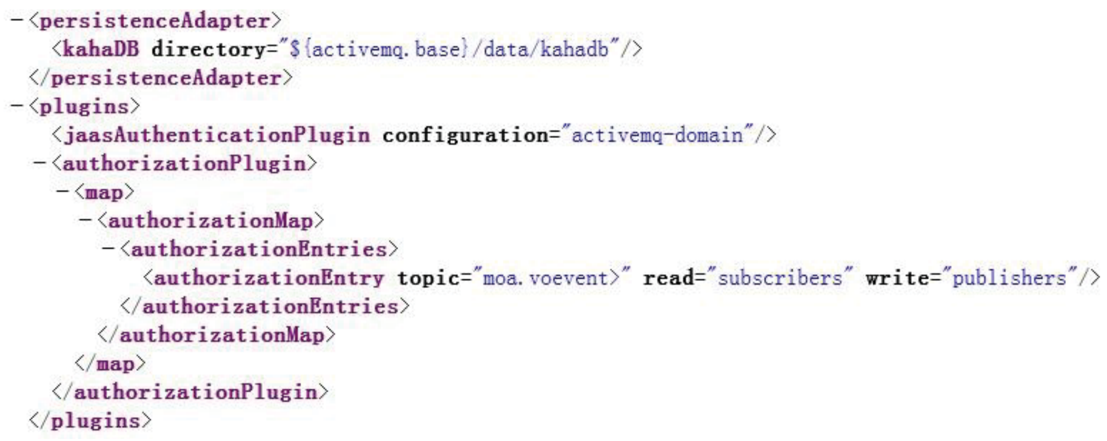


Figure 25: Example data format of `user.properties` and `group.properties` files. The role name in the `group.properties` file can be any name given by the developer, rather than being limited to publishers and subscribers as the example above. However, the user name in `group.properties` file must correspond to the user name in the `user.properties` file.

Even though the data are stored as plain text format in both the "`user.properties`" and "`group.properties`" files, these two files will only be able to be read by the message oriented middleware internally. In other words, these two files are not associated with or exposed to any external applications. Thus, the data stored in both files will not be steal by hackers through SQL injection attacks.

After both `user.properties` and `group.properties` files have been created, the second step is adding the JAAS authentication plugin into the ActiveMQ configuration file. The JAAS authentication plugin is written in XML format and example codes are shown in Figure 26.



```
- <persistenceAdapter>
    <kahaDB directory="${activemq.base}/data/kahadb"/>
  </persistenceAdapter>
- <plugins>
    <jaasAuthenticationPlugin configuration="activemq-domain"/>
  - <authorizationPlugin>
    - <map>
      - <authorizationMap>
        - <authorizationEntries>
            <authorizationEntry topic="moa.voevent>" read="subscribers" write="publishers"/>
          </authorizationEntries>
        </authorizationMap>
      </map>
    </authorizationPlugin>
  </plugins>
```

Figure 26: The code fragments of the JAAS authentication plugin in the ActiveMQ configuration file.

Each `<authorizationEntry>` tag represents a single topic in the JAAS plugin; the number of `<authorizationEntry>` tags must correspond to the number of topics that are created in the ActiveMQ. For example, our ActiveMQ only has a single topic (`moa.voevent`) for disseminating new transient information, so there is only one `<authorizationEntry>` tag in our ActiveMQ configuration file. If the ActiveMQ contains more than one topic, more `<authorizationEntry>` tags are required in the ActiveMQ configuration file.

Moreover, the value of attribute `read` indicates the roles that are able to subscribe and consume the messaging from this topic. Similarly, the value of attribute `write` indicates the roles that are able to publish messaging to this topic. In this example, the role of user A and user B is allowed to publish messaging to the `moa.voevent` topic, but the role of user C is only allowed to subscribe and consume messaging from the `moa.voevent` topic. Note that any of the authentication plugins (simple authentication or JAAS authentication) must be written under the `<presistenceAdapter>` tag due to the element structure that is predefined in the ActiveMQ XML schema file.

Indeed, the JAAS also allows for storage and obtaining of user name and password from databases through a JDBC (Java Database Connectivity) connection [257] [258]. However, the processes of using databases for storing and obtaining user authentication details are much more complicated than the properties files approach, and using the properties files approach for user authentication is good enough for our systems.

The goal of topic protection can be very easily achieved through JAAS due to the fact that it offers very powerful and flexible mechanisms for user authentication; thus, we use the JAAS mechanism for user authentication in our ActiveMQ server.

Automatic User Registration System

The second stage of the research is almost complete with the implementation of the security mechanism. However, there is still one issue to be resolved, which is user registration. This is because ActiveMQ does not provide any automatic user registration features by default, so registering a new user account in ActiveMQ requires a manual operation. This means that if a new user wants to use any features of our ActiveMQ server, he or she needs to send a registration request email with his selected user name and password to MOA. The MOA administrator then needs to add this user name and password into both the "`user.properties`" and "`group.properties`" files manually to complete the new user account registration process. This manual user registration is a very inefficient operation and it also requires that the MOA administrator checks their email frequently to see whether new registration requests have been received.

Thus, we have developed an automatic registration system to make user registration a fully automated operation without any manual intervention being required.

The automatic user register system is implemented based on a three layer structure and consisting of three different components, as outlined below:

1. The first component is the web page, which is also called the view layer or front end layer. The main responsibility of this component is providing a GUI (Graphic User Interface) for user registration. The Webpage is implemented based on the JSP (Java Server Pages) technique; it contains a simple web form and the end user needs to fill out this form, providing the necessary information for creating a new user account in ActiveMQ, such as a user name and password. In fact, there is one potential security issue that may occur in this front end layer, which is that malicious hackers can write an automatic registration application to register massive amounts of users' data to cause the system to break down. Thus, a protection mechanism needs to be applied to the front end layer in order to prevent this security issue.

   The most common way is using a validation code on the registration page; when a user attempts to registers an account in the system, they need to input the validation code before submitting the registration information to the back end server. If the validation code input by the user matches the validation code displayed on the registration page, the user registration information will be sent to the back end server, otherwise the web page will not perform any further operation until the correct validation codes has been input by the user. Note that the validation codes should be generated randomly each time, so the hackers cannot guess what the next validation code will be.

2. The second component is the database, which is also called the model layer. The main reason for involving the database is for persistence user data of our automatic registration system. As mentioned in the topic protection section,

duplicate user names are not acceptable in the ActiveMQ security domain, but because all user account information (user name and password) are stored in plain text format in the "`user.properties`" file, the only way of checking for duplication of user names is to compare the newly registered user name with each user name that already exists in the "`user.properties`" file, such as in the following pseudo code:

```
//Reading all user names from "user.properties" file
for (int i = 0; i < user name count; i++)
{
  if (new register user name.equal(existed user name))
  {
      return user name is existed;
  }
}
//Writing new register user name into "user.properties" file
```

However, while large numbers of user names exist in the "user.properties" file, the performance of the above algorithm will decrease linearly. Through using the database, this performance issue can be effectively resolved, because the database provides an optimisation query engine that will guarantee the performance of mining particular data from a large dataset.

The database design of the automatic user register system is very simple, containing a single table to record all user account information and the table schema, as illustrated below:

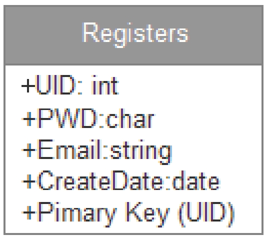


Figure 27: User registration table schema.

The column UID and PWD are used to record user names and passwords. The email column records the email address of the end user. This column is designed for finding password features; if end users forget their password, they can use the find password feature to find their password, which password will automatically be sent to their email address. The `CreateDate` column is only designed for management purposes; with this column, the system administrator will be aware of when the end user registered in the system. In fact, performance improvement is not the only advantage brought by integrating the database; the database also helps reduce the complexity of development. For example: checking for duplicate user names can be done through a simple SQL command "`select * from register where UID = 'new register user name'`", instead of using the complex comparison algorithm discussed above.

However, the database may get a SQL injection attack if we directly use said select statement for querying the database. Thus, we use the prepared statement [259] method to deal with the SQL command described above. There are two benefits that can be obtained by using the prepared statement method to access the database. First, the prepared statement will precompile the SQL statement before sending the SQL statement to the database, and the database can directly run the SQL statement compiled by the prepared statement without compiling it first. Once a SQL statement has been compiled by prepared statement and sent to the database, it will be stored in the database server. In other words, the same SQL statement can be executed multiple times and only needs to compile once, rather than compiling it every time it needs to be executed. Therefore, the execution time of a SQL command is reduced significantly. Second, the prepared statement allows the use of parameters in the SQL query command; with a parametric command, all the parameters passed as part of the place-holder will be escaped automatically by the JDBC driver, so the well-known SQL injection attack can be avoided.

Furthermore, the passwords should not be stored as plain text in the database, because if the database is hacked, the hackers can obtain all user names and passwords stored in the database. Afterwards, the hackers can login to the message oriented middleware and publish malicious messages to subscribers.
Therefore, passwords should be encrypted before being stored into the database.
In general, there are three common approaches that can be utilised for password encryption. The first one is to store the hash codes of the password into the database; this is simple and the fastest method of password encryption. However, with this approach, the hashed password can still be very easily decoded through comparison with a hash table (there are plenty of hash tables available online).
The second approach is to combine the password with a salt (the salt can be any string value), so every password will have extra data before hashing; however. the same salt is used in this approach for all of the passwords, which means that the extra data of all passwords are the same. Using this scheme, the hacker can only get the password through bruteforce hashes, so it will take some time to decode the passwords. However, as soon as one password is cracked, the passwords of all other system users who use the same password will also be revealed.

The third approach is a variation from the second approach; this approach combines the password with a random salt, which means that each password has different extra data. Therefore, even if all system users have the same passwords, the hash codes will still be completely different. In this case, if a hacker wants to steal all users' passwords from the database, they will need to bruteforce every password.

Thus, the password should be encrypted with a random salt before being stored in the database in order to increase the difficulty of bruteforce and stealing the passwords from the database.

3. The last component is the servlet; it acts as an intermediary layer between the Website and the database, and has responses to deal with all registration processes

logic, so this layer is also termed the controller layer. When the servlet receives a new registration request from the website, it will determine whether or not the newly registered username already exists through using the SQL command mentioned above. If the newly registered username already exists in the database, the servlet will return a username exists message to the website to inform the end user that their selected username is already registered by another end user. On the other hand, if the newly registered username has not been selected by a previous user, the new username will be stored into the database through the use of the SQL insert command. After the username has been successful stored into the database, the next operation performed by the servlet is storing the newly registered username into the ActiveMQ security domain. As mentioned above, the ActiveMQ security domain is related to the "`user.properties`" file and the "`group.properties`" file, so the newly registered username needs to be stored into both files as well.

```java
private boolean writingToActiveMQSecurityDomain(
                String groupFilePath, String usersFilePath,
                String userName, String password){
  boolean completedWriting = false;
  try {
    //read values of both files and store in properties object
    Properties groupProperties = readProperties(groupFilePath);
    Properties userProperties = readProperties(usersFilePath);
    //get all subscriber usernames existed in group.properties
file
    String propertieValue = getGroupPropertiesValue("subscribers",
                                groupProperties);
    //append new registering username
    propertieValue = propertieValue.concat ("," + userName);
    //storing new username list in groupProperties object
    groupProperties.setProperty("subscribers", propertieValue);
    //storing new registering username in userProperties object
    userProperties.setProperty(userName, password);
    //Writing new user data into two properties files
    completedWriting = storeProperties(groupFilePath,
                          usersFilePath, groupProperties,
                          userProperties);
  } catch (Exception ex) {
      Return false;
  }
  //A true value will be returned to notify servlet that new
  //new username has been successful stored into both files.
  return completedWriting;
}
```

Code example 1: Registering new user to ActiveMQ security domain files.

```
private boolean storeProperties(String groupFilePath,
String usersFilePath, Properties groupPropertie,
Properties usersPropertie)
{
  FileOutputStream outputFile = null;
  try {
   //IO operations
    outputFile = new FileOutputStream(groupFilePath);
    groupPropertie.store(outputFile, null);
    outputFile.close();
    outputFile = new FileOutputStream(usersFilePath);
    usersPropertie.store(outputFile, null);
    outputFile.close();
   } catch (Exception ex) {
    //the rollback operation responses to removes new registering
    //while one of IO operation is failure.
    rollback();
    return false;
}
```

Code example 2: File IO operations, writing new username and password into both “`user.properties`” and “`group.properties`” files.

The storage processes are done by two file IO (Input/Output) operations, and can be programmed as in the code example 1 and 2 above, and the servlet will return a successful registration message to the website after all the storage processes have been completed.

Our automatic registration system is implemented through the JSP, MySQL and the servlet framework, but these three techniques can be replaced by other similar frameworks, such as ASP.NET or PHP.

In fact, this transient architecture is still able to work without an automatic registration system, but we strongly recommend implementing this automatic registration system, because it will increase the automation nature of the whole transient notification architecture. The flow chart diagram of the servlet is illustrates in Figure 28.

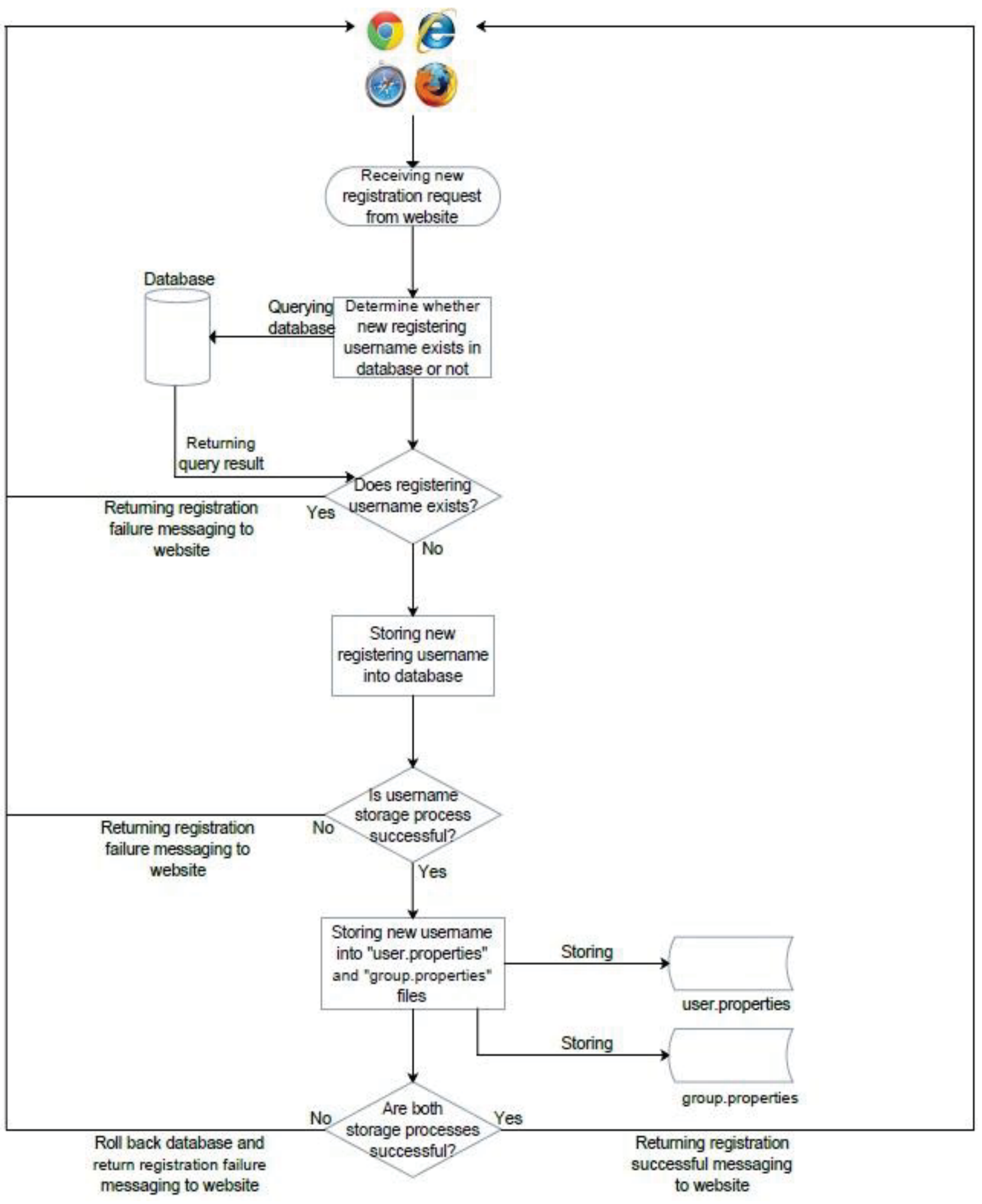

Receiving new registration request from website
Database
Querying database
Determine whether new registering username exists in database or not
Returning query result
Does registering username exists?
Returning registration failure messaging to website
Yes
No
Storing new registering username into database
Is username storage process successful?
Returning registration failure messaging to website
No
Yes
Storing new username into "user.properties" and "group.properties" files
Storing
user.properties
Storing
group.properties
Are both storage processes successful?
No
Yes
Roll back database and return registration failure messaging to website
Returning registration successful messaging to website

Figure 28: The complete workflow of the servlet component, this diagram is produces by Edraw Max.

Once all of the components described in this section have been set up, the new transient notification architecture is ready for public use, and stage two of this research is fully completed.

In the following two sections, we will describe how to publish the VOEvent packet and receive the VOEvent packet in this transient notification architecture, which is also referred to as the client side applications development.

4.4 New Transient Information Publishing Pipeline

The new transient information publishing pipeline will execute when a new transient event needs to be alerted to the microlensing communities; this publishing pipeline mainly consists of two separate applications, which are the VOEvent packet generator and the transient publisher.

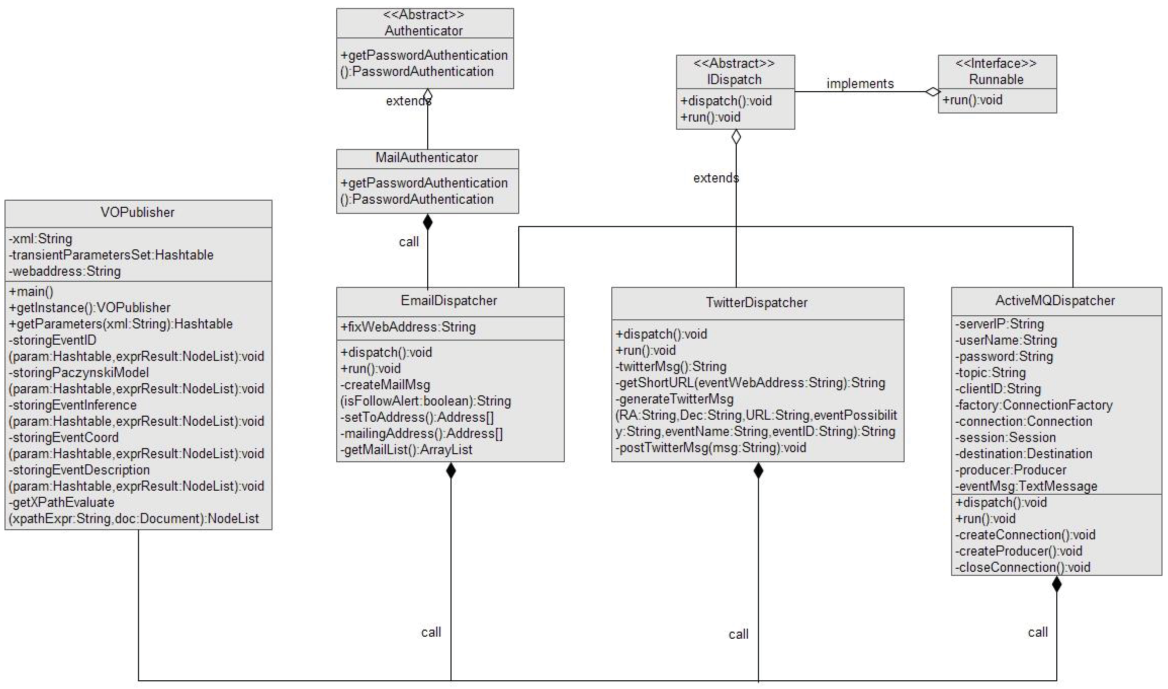
The VOEvent packet generator is not the subject of our research, so we will not discuss its implementation in much detail, but the end result of this VOEvent packet generator is a VOEvent packet string. After the VOEvent packet string has been generated by the VOEvent packet generator, the VOEvent packet generator will execute the transient dispatcher application and feed the VOEvent packet string as the input parameter, then passing full control to the transient dispatcher application.

The transient publisher application will perform three different publishing tasks after receiving the VOEvent packet string from the VOEvent packet generator. The concept of this transient publisher application is the main aspect that we will discuss in this section.

4.4.1 The Concept of the New Transient Publisher Application

The new transient dispatcher is an integration application and is implemented in Java. It will publish new transient information to three different platforms (email, Twitter and message oriented middleware), rather than just publishing the VOEvent packet string that is received from the VOEvent packet generator to the ActiveMQ. This multi-platform publishing concept allows end users to use their preferred method of receiving new transient notifications instead of being limited to only receiving new transient notifications from ActiveMQ.

Each publishing task can be seen as an individual application; however, if we develop an application for each publishing task, there will be a total of three different applications to perform similar publishing tasks. The complexity of the publishing pipeline will be increased and the extra lines of code will be generated in the VOEvent packet generator as well, because the VOEvent packet generator will need to call each publishing task application once in order to publish new transient information to the three different platforms described above. Thus, we integrated these three different publishing tasks into one single application, so the VOEvent packet generator only needs to call one application to perform three different publishing tasks, rather than calling three different applications. Additionally, integrating three different publishing tasks into a single application will reduce the complexity of code maintenance in the future.



<<Abstract>> Authenticator
+getPasswordAuthentication ():PasswordAuthentication
extends
MailAuthenticator
+getPasswordAuthentication ():PasswordAuthentication
call
<<Abstract>> IDispatch
+dispatch():void
+run():void
implements
<<Interface>> Runnable
+run():void
extends
VOPublisher
-xml:String
-transientParametersSet:Hashtable
-webaddress:String
+main()
+getInstance():VOPublisher
+getParameters(xml:String):Hashtable
-storingEventID (param:Hashtable,exprResult:NodeList):void
-storingPaczynskiModel (param:Hashtable,exprResult:NodeList):void
-storingEventInference (param:Hashtable,exprResult:NodeList):void
-storingEventCoord (param:Hashtable,exprResult:NodeList):void
-storingEventDescription (param:Hashtable,exprResult:NodeList):void
-getXPathEvaluate (xpathExpr:String,doc:Document):NodeList
EmailDispatcher
+fixWebAddress:String
+dispatch():void
+run():void
-createMailMsg (isFollowAlert:boolean):String
-setToAddress():Address[]
-mailingAddress():Address[]
-getMailList():ArrayList
TwitterDispatcher
+dispatch():void
+run():void
-twitterMsg():String
-getShortURL(eventWebAddress:String):String
-generateTwitterMsg (RA:String,Dec:String,URL:String,eventPossibility:String,eventName:String,eventID:String):String
-postTwitterMsg(msg:String):void
ActiveMQDispatcher
-serverIP:String
-userName:String
-password:String
-topic:String
-clientID:String
-factory:ConnectionFactory
-connection:Connection
-session:Session
-destination:Destination
-producer:Producer
-eventMsg:TextMessage
+dispatch():void
+run():void
-createConnection():void
-createProducer():void
-closeConnection():void
call
call
call

Figure 29: Class diagram of new transient publisher application.

As we know, when a program starts up, a thread begins running immediately, which is usually called the main thread of the program. By default, all methods defined in the program codes will be executed one by one in the main thread. However, executing all methods in the main thread might raise a potential problem, because in a single thread program context, the latter methods will only be executed when the former methods are completed. In other words, if any problem occurs in one of the publishing tasks, all of the latter publishing tasks will be impacted. For example: the email publishing task is the first publishing task defined in the publisher codes, so it will be first executed by the program, but if there are any problems (i.e: it cannot connect to the SMTP server, there is an invalid username, or password for the email account) occur in this task, the Twitter and ActiveMQ publishing tasks can only be executed until the runtime exception is thrown up by the email publishing task.

In the worse case scenario, the publisher application will stop working, so end users might miss the chance to observe interesting transients. Therefore, we have utilised multi-threading in the publisher application to avoid this issue. With multi-threading, all of the three publishing tasks are performed in parallel and each publishing task uses a separate thread, so even if a problem occurs in one publishing task, the rest of the publishing tasks are still able to publish the new transient information to their corresponding platforms. The multi-threading feature can be implemented as the following codes:

```
//Create instance of each publishing task
Object o[] = {new ActiveMQDispatcher(),
              new TwitterDispatcher(),
              new EMailDispatcher()};
//Performing publishing tasks
for (int i = 0; i < o.length; i++)
{
    IDispatcher dispatch = (IDispatcher) o[i];
    //Creating a thread for each publishing task
    new Thread(dispatch).start();
}
```

Code example 3: Codes for implementing the multi-threading feature for dispatcher application

Moreover, the VOEvent packet is a machine readable format rather than human readable format, so publishing the entire VOEvent packet string to both email and Twitter (Twitter also has characters limit for each tweet, so it is not possible to publish the whole VOEvent packet in one tweet) platforms is not a sound idea. Therefore, we need to extract important parameters' values of the event from the input VOEvent packet string in the main entry class of the publisher application (in our case, the class name of main entry is called VOPublisher), and store all the extracted parameters' values into a hashtable called the `transientParametersSet`.

Extracting parameter values from a VOEvent packet is very simple; we present code examples to extract the name of a transient event from a VOEvent packet string and store this name in the Hashtable as a parameter value extraction example. The detailed steps are outlined below:

1. The transient event name consists of the field name, filter colour, chip and sequence number. All of these parameters are stored as attribute values of the nested nodes `<Param>` under the `<Discovery_Image>` node. Therefore, the first step uses XPath to obtain all the nested nodes of `<Discovery_Image>` and stores these nested nodes in a `NodeList` object. The example code is as shown below:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
StringReader st = new StringReader(voevent);
//Parse VOEvent packet string in an Document object
Document document = db.parse(new InputSource(st));
//XPath for getting all nested nodes of Discovery_Image node
String xpath = "//What/Group[@name='Discovery_Image']/*";
NodeList exprResult = null;
try {
  XPath xpath = XPathFactory.newInstance().newXPath();
  XPathExpression expr = xpath.compile(xpath);
  //evaluate method return xpath result as object
  Object result = expr.evaluate(doc, XPathConstants.NODESET);
  //typecasting to NodeList type in order to stores
  //xpath result in a NodeList
  exprResult = (NodeList)result;
} catch (Exception ex) {
  System.out.println(ex.getMessage());
}
```

Code example 4: Extracts all nested nodes of `<Discovery_Image>`

2. After all of the nest nodes are stored in the `NodeList` object, we need to use the `getAttributes` method to obtain the attributes of the `<Param>` nodes. The `getAttributes` method will return all attributes of a single node. Moreover,

the `<Discovery_Image>` node contains lots of nest nodes, so we must use a `for` loop to go through the entire `NodeList` in order to obtain all of the parameter values that we want. The code example 5 illustrates processes of constructs transient event name.

```
//necessary parameter name for construct transient event name
String field = "";
String colour = "";
String chip = "";
String seq = "";
String eventName = "";

//go through entire NodeList
for (int i = 0; i < exprResult.getLength(); i++)
{
   //get all attributes of a single node
   //and stores in a NamedNodeMap object
   NamedNodeMap nnm = exprResult.item(i).getAttributes();
   if (nnm.getLength() != 0)
   {
       //Param node contains 3 attributes, first attribute
       //describes parameter name
       if (nnm.item(0).getTextContent().equals("field")) {
           //stores parameter value in corresponding variable
           field = nnm.item(2).getTextContent();
       }
          //stores other parameter values in variables
   }
 }
   //construct transient event name
   eventName = field + "-" + colour + "-" + chip + "-" + seq;
   //stores in transient event name in hastable
   transientParametersSet.put("EventName", eventName);
```

Code example 5: Codes that extracts attributes of `<Param>` node, constructs transient event name and stores in the `transientParameterSet` hashtable

Once the above codes have been executed, the transient event name will be stored in `transientParametersSet`, and we can use the same method to extract other necessary parameters from the VOEvent packet string and store them in the `transientParametersSet`. After all of the necessary parameters have been stored, the `transientParametersSet` Hashtable is ready for use by different publishing tasks.

4.4.2 Email Publishing Task

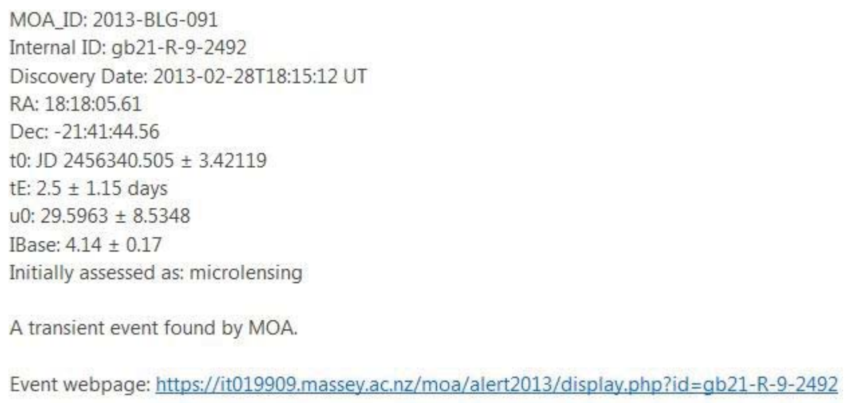
Some end users might still prefer to receive new transient information through email, therefore, we decide to retain the traditional email notification method in our transient publishing pipeline.

Implementing an email publishing task is not complex and only requires three steps. The first step is reading all the email addresses from the mailing list file (in our experimental system, the mailing list is a text based file with a .txt extension), and storing them into an array as a sending list. The reading and storage operations can be implemented through using a file I/O (input/output) function, using the codes shown below:

```
ArrayList<String> mailList = new ArrayList<String>();
File f = new File("/home/iabond/MOA/Dispatcher/mails.txt");
String tmp;
BufferedReader br = new BufferedReader(new InputStreamReader
                          (new FileInputStream(f)));
//reading mailing list file line by line
while ((tmp = br.readLine()) != null)
{
  //Any email address in mailing list file
  //that start with # will be skip
  if (tmp.indexOf("#") == -1)
  {
     //adding to sending list
     mailList.add(tmp);
  }
}
return mailList;
```

Code example 6: Reading the mailing list from a txt file

Second, after all of the email addresses have been stored in the sending list, the contents of the notification email message need to be created, and the contents should describe the new transient event in a brief way. Thus, we only include eight major transient parameters in email contents, which are the name of the transient event, a description, external ID, internal ID, sky coordination (RA and DEC), model parameters, initial nature (transient inference) and the link to the event profile webpage. The values of these parameters can be acquired from the `transientParametersSet` hashtable, which is created in the main entry class. With these eight parameters, the end users will have some basic idea as to whether or not the new transient event is of interest to them. If it is of interest to them, they can click the link in the notification email to access the full profile of the event for advanced analysis.



MOA_ID: 2013-BLG-091
Internal ID: gb21-R-9-2492
Discovery Date: 2013-02-28T18:15:12 UT
RA: 18:18:05.61
Dec: -21:41:44.56
t0: JD 2456340.505 ± 3.42119
tE: 2.5 ± 1.15 days
u0: 29.5963 ± 8.5348
IBase: 4.14 ± 0.17
Initially assessed as: microlensing

A transient event found by MOA.

Event webpage: https://it019909.massey.ac.nz/moa/alert2013/display.php?id=gb21-R-9-2492

Figure 30: Example of notification email contents.

The final step is to send out a notification email to all recipients in the sending list. The email sending operation can be performed simply with Email API under a `javax.mail` name space, so we will not discuss the implementation details here. After all of the notification emails have been sent out, the email publishing task is completed.

Performance limitation of Email Publishing Task

We cannot control the performance of the email publishing task, because when an email has been send, it will be sent to an SMTP server. Afterwards, the SMTP server will examine the email header and relay the email to the servers that holds the email services of the recipients. If the recipients belong to same email domain as the email sender, they can receive the notification email very quickly, because the SMTP server does not need to search and contact other server; this is also referred to as local delivery [260] [261].

However, in our system, most of the email addresses of recipients belong in different email domains; in this case, the SMTP server will need to search and obtain IP addresses of each email domain contained in the email header from the DNS server, and then connect and relay the email to the corresponding email servers. In other words, the performance of the email delivery is dependent on how the SMTP server transfers the email, and we have no idea whether or not the notification email is successfully delivered to the intended recipients. In the worst case scenario, if the problem occurs in the SMTP server, the notification email might not be delivered to the end users at all.

4.4.3 Twitter Publishing Task

Social website platforms have become very popular platforms for people acquiring information; many people visit social website platforms many times each day, so it is worthwhile to publish new transient information on social website platforms. Thus, we implement a Twitter publishing task in our publisher application; this publishing task will post a tweet message that describes new transient information on the MOA twitter account, and all MOA followers will receive the new transient information from either the Twitter website or the Twitter mobile application immediately. In fact, Twitter works in quite a different way from email. When we send a tweet message, the tweet message will reach the Twitter server as the first stage, and all the remaining operations (find user account, update new tweet on corresponding user account, list new tweet on follower account, etc.) are done in the Twitter server, rather than forwarding the tweet message to some other servers as with email. Furthermore, when a new tweet message has been updated to the MOA twitter account, the Twitter server will respond by sending an error message to the Twitter publishing task when the tweet message fails to update to the MOA twitter account, so we will know whether or not followers can see the new transient information.

The first step in implementing the Twitter publishing task is registering the Twitter developer account to obtain Twitter Rest [146] API access keys, which are the `OAuthConsumerKey`, `OAuthConsumerSecret`, `OAuthAccessToken` and `OAuthAccessTokenSecret`. These four keys are very important and are unique for each developer account. Without these four keys, we are not able to post "tweet" messages on Twitter from our publisher application.

After acquiring the Twitter API access keys, the second step is designing what type of transient information should be presented in the tweet message. In fact, the new transient information presented in the tweet message cannot be same as in the email message, because Twitter has a character limit (140 characters) for each tweet message, so the tweet message should describe the new transient information in a much briefer way than the email message. After discussion, we decided to present the external ID of the new transient, sky coordination (RA and Dec), initial nature and web URL of the new transient profile in the tweet message, because these pieces of information can provide a basic picture of the new transient to end users. If the end users are interested in the new transient, they can access the full transient profile through clicking on the web URL in the tweet message.

Note that we cannot present the full web URL of the transient profile in the tweet message, because this will cause the message to go over the character limit. Thus, we need to transform the full web URL to a short URL [147] format. The web URL shortened operation can be done through using the `bit.ly` service, which is a URL shortening service; it provides the ability to transform a web URL from the full URL format to a short URL format, and it is the default URL shortening service used by Twitter. Example codes for transfer full URL to short URL via using `bit.ly` service are as shown below:

```
HttpClient httpclient = new HttpClient();
//bitly service API address
HttpMethod method = new GetMethod("http://api.bit.ly/shorten");
method.setQueryString(new NameValuePair[] {
                          new NameValuePair("version", "2.0.1"),
                          new NameValuePair("longUrl",
                                            eventWebAddress),
                          new NameValuePair("login","user account"),
                          new NameValuePair("apiKey",
                                            "API access key"),
                           //bit service supports xml and
                          //json as return format
                          new NameValuePair("format", "xml")}
                          );
 httpclient.executeMethod(method);
```

Code example 7: using `HttpClient` to communicate `bit.ly` REST API in order to transform full web URL to short URL format.

To using the `bit.ly` service, we need to register a `bit.ly` development account in order to obtain a service API access key, and then we can access the URL shortening API by using the http communication feature provided by the programming language (in our example this is `HttpClient`).

After the above codes have been executed, the `bit.ly` service will return an XML string as the response result, and we can retrieve the short URL address from the XML string through utilising XML DOM (Document Object Model). The short URL retrieval process can be programmed as code example 8:

```
String responseXML = method.getResponseBodyAsString();
if (responseXML != null)
{
   DocumentBuilderFactory dbf =
   DocumentBuilderFactory.newInstance();
   DocumentBuilder db = dbf.newDocumentBuilder();
   StringReader st = new StringReader(responseXML);
   Document d = db.parse(new InputSource(st));
   NodeList nl = d.getElementsByTagName("shortUrl");
    if (nl != null)
   {
       Node n = nl.item(0);
       shortURL = n.getTextContent();
   }
}
```

Code example 8: Get the response from `bit.ly` and extracts the shortened URL format from the XML document.

Once the short URL has been retrieved, all of the necessary information for the tweet message is ready, and the last pieces of code for this publishing task are posting the tweet message on the MOA Twitter account.

To post a tweet message on Twitter, the Twitter library needs to be downloaded and implemented in the publisher application. Twitter provides its library in many programming languages for developers, but we only use the Java version of the Twitter library in our example, because our publisher application is implemented based on Java.

Once the Twitter library has been successfully included in the publisher application, we are able to post a tweet message on Twitter through the use of the following codes:

```
ConfigurationBuilder cb = new ConfigurationBuilder();
//Twitter API access keys obtain in the step 1
cb.setOAuthConsumerKey("consumerKey ");
cb.setOAuthConsumerSecret("ConsumerSecret");
cb.setOAuthAccessToken("AccessToken");
cb.setOAuthAccessTokenSecret("TokenSecret")
//Create Twitter factory instance, the Twitter factory
//is class provided by the Twitter library
TwitterFactory factor = new TwitterFactory(cb.build());
Twitter twitter = factor.getInstance();
//Posting twitee message on Twitter
twitter.updateStatus(msg);
```

Code example 9: Posing new tweet about new transient information on MOA's Twitter account. The variable "`msg`" in the updateStatus method is the actual message that needs to be posted on Twitter; it can be any string value.

If above codes are successfully executed, the new tweet message will be sent to the Twitter server after the updateStatus function is completed, and the time taken for the new tweet message displayed on the twitter account is fully dependent on the network traffic of both the sender side and the Twitter server side.

Performance limitation of Twitter Publishing Task

The performance limitation of the Twitter publishing task is similar to the email publishing task; if both publisher application side and Twitter server side has a good network connection environment, the new transient information will be updated on MOA's twitter account immediately after the Twitter publishing task is completed.

However, if the Twitter publishing task has the same network performance issue as the email publishing task, this network issue will lead to there being a delay in the updating of new transient information on MOA's twitter account. In fact, a network latency problem exists in all of the distribution systems and there is no way to avoid it.

In fact, email and Twitter present no problems for event rates in the current transient detection projects, but these two approaches are clearly unsuitable for LSST. However, LSST could post those very important transient discoveries on Twitter, or send email notifications to all interested recipients.

4.4.4 ActiveMQ Publishing Task

The ActiveMQ publishing task is one of the crucial parts of our research, because it plays such an important role in the new transient notification architecture that we have proposed. Under the Publisher/Subscriber (Pub/Sub) context this role is publishing the VOEvent packet to the topic (moa.voevent) that has been created in the ActiveMQ.

The ActiveMQ publishing task will publish the entire VOEvent packet that is inputted from the VOEGenerator to the "`moa.voevent`" topic, rather than using the `transientParametersSet` hashtable as the email publishing task and Twitter publishing task did.

In fact, the VOEvent packet is the major data that MOA wants to deliver to end users, because the VOEvent packet is typically designed for new transient notification, and it contains all of the metadata about the new transient. End users can utilise the VOEvent packet to do lots of different operations (such as driving robotic telescopes, archiving searches, or a transient analysis processes).

Implementing the ActiveMQ publishing task in Java requires the use of the JMS API under the `javax.jms` namespace. The JMS API is a fully implemented JMS specification and it can be used for communicating with every message oriented middleware that implements the JMS specification.

Thus, the ActiveMQ publishing task can be used for publishing the VOEvent packet to any JMS based message oriented middleware, rather than only working with the ActiveMQ server.

The VOEvent packet publishing process can be divided into three steps; establishing a connection with ActiveMQ, publishing messaging to the ActiveMQ server, and the close connection between the publishing task and the ActiveMQ server. Each step will be explained in detail below and illustrated using example codes.

Two JMS objects need to be created in order to establish a connection between the ActiveMQ publishing task and the ActiveMQ server; these are `connectionFactory` object, and `connection` object. The `connectionFactory` object encapsulates a set of connection configuration parameters that are necessary to establish a connection with the ActiveMQ server, including a login user name, login password and the IP address of ActiveMQ server. After the connectionFactory object has been created, the `connection` object will use it and invoke the `start()` method to establish a new connection with the ActiveMQ. The following code shows the establishment of a connection with ActiveMQ in the publishing task.

```
ConnectionFactory factory = new ConnectionFactory(username,
                                  password, serverIP);
Connection connection = factory.createConnection();
connection.start();
```

Code example 10: Creates `ConnectionFactory` object and establish a new connection with ActiveMQ

Once the publishing task has successfully established a connection with ActiveMQ, the second step is publishing the VOEvent packet to the ActiveMQ server. Four objects need to be created in order to complete this step. The first is the `Session` object. The session is a single-threaded context for producing and consuming messages, and it also provides a transactional context with which to group a set of sends and receives into an atomic unit of work. We need to use the method provided by the `Session` object to create message producers, message consumers and messages.

The second object we need to create is the `Destination` object. The destination is the object that is used to specify the target of the messages it produces and the source of the messages it consumes. In our experimental architecture, the target of the messages is the “`moa.voevent`” topic we created in the previous section.

The third object that needs to be created is the message object, because JMS uses the message object to store the message that needs to be sent. There are in total five different types of message objects that can be created through using the JMS API, and these five different types of message objects can be used for storing different types of data in the messages (i.e., we can deliver image files to end users through converting the image file to binary codes and storing it in a binary message object). The type of message object we use for storing the VOEvent packet is `TextMessage`. The `TextMessage` object is typically designed for storing and dealing with pure string messages. As is well-known, the VOEvent packet is based on an XML document and an XML document consists of a pure string, so the `TextMessage` object is adequate to fulfill our requirements.

After the VOEvent packet has been stored in the `TextMessage` object, the last object that needs to be created is the `MessageProducer` object. The `MessageProducer` is an object that is created by a `Session` and used for sending messages to a topic. Please note that we also need to define the message delivery mode before sending a message to a topic, otherwise `MessageProducer` will send messages through using default delivery mode (Non-Persistent).

There are two delivery modes supported by the JMS API; Non-Persistent, and Persistent. The Non-Persistent delivery mode is the lowest-overhead delivery mode because it does not require that the message be logged to stable storage. However, if any errors occur on the message oriented middleware, the messages that come from the message sender will be lost and never delivered to the receivers.

On the other hand, the Persistent delivery mode performs a very different operation when compared with the Non-Persistent delivery mode. In the Persistent delivery mode, the message oriented middleware will log a message to stable storage, guaranteeing that the message be delivered once, and only once, to the message receiver. Any message that is marked as Persistent delivery will not be lost unless hardware storage failure occurs. The Persistent delivery mode is very useful when the publisher wants to deliver every published message to the message receiver without missing.

We have used the Persistent delivery mode in our architecture, because we need to ensure delivery of every VOEvent packet to all end users without any being lost.

After the message delivery mode has been defined, we can than invoke the `send` method to send a message to the topic. The example codes below show how to create and use `MessageProducer` to send a message to the topic.

```
Session session = connection.createSession(false,
                  Session.AUTO_ACKNOWLEDGE);
//Specifing message destination
Destination destination = session.createTopic("moa.voevent");
//Creating message producer object
MessageProducer producer = session.createProducer(destination);
//Setup delivery mode as persistent
producer.setDeliveryMode(DeliveryMode.PERSISTENT);
//Creating TextMessage object
TextMessage eventMsg = session.createTextMessage();
eventMsg.setText(VOEvent packet string);
//Sends VOEvent packet to the topic
producer.send(eventMsg);
```

Code example 11: Sends a new VOEvent packet string to "moa.voevent" topic through using persistent delivery mode

Once a VOEvent packet has been successfully sent to the topic in the ActiveMQ server, the final step involves invoking the `close` method provided by the `Connection` object to disconnect the connection between the ActiveMQ publishing task and the ActiveMQ server, in order to save the resources of the ActiveMQ server and host machine. The example code of the close connection operation is as shown below:

```
connection.close();
```

Code example 12: Disconnects dispatcher application from ActiveMQ server

After the code `connection.close()` has been executed, the entire ActiveMQ publishing task is completed, the VOEvent packet should reach the "`moa.voevent`" topic and the ActiveMQ server will forward the VOEvent packet to all online topic subscribers immediately (topic subscribers will be described in detail in the next section).

4.5 Subscriber Application for Receiving New Transients' Information

The subscriber application is another important component of our new transient notification architecture, because the subscriber application responds to subscribers, receives and processes VOEvent packets that have been published to the "`moa.voevent`" topic, and plays the `Subscriber` role under the Pub/Sub context.

The implementation processes of the subscriber application are very similar to the publishing task. First, a `Connection` object needs to be created for establishing a connection with the ActiveMQ server. However, this connection is a persistent connection, which means that the connection between the subscriber application and the ActiveMQ server will not disconnect until the end user initiative close subscriber application or `disconnect()` method has been invoked. After this we also need to create the `Destination` object for specifying the topic that needs to be subscribed to.

Second, a subscriber object needs to be created for the topic subscription. JMS provides two different types of subscriber objects; the `MessageConsumer` object, and the `TopicSubscriber` object. These two subscriber objects correspond to two different topic subscription modes, which are non-durable topic subscription and durable topic subscription. With the non-durable subscription mode, the subscriber application can only receive messages that are published while it is active. This mode can be made active through using the `MessageConsumer` object and invokes the `session.createConsumer()` method.

The durable subscription mode works in the opposite way to the non-durable subscriber mode. With durable subscription mode, the subscriber application will receive all the messages published on a topic, including those that are published while the subscriber application is inactive; the messages published during the inactive period will be delivered to the subscriber application immediately when it is next active. In addition, the JMS specification does not state any limits of durable subscribers for a single message topic [262], so the "*moa.voevent*" topic can be subscribed to by as many subscriber applications as necessary. The durable subscription mode can be made active through using the *TopicSubscriber* object and invoking the *session.createDurableSubscriber()* method.

In our experimental application, we are using a durable subscription mode to subscribe to the "`moa.voevent`" topic, because this mode will ensure that our subscriber application receives all of the published VOEvent packets from the "`moa.voevent`" topic.

Third, it is necessary that a `MessageListener` is registered with the subscriber object in order to receive messages from the topic. The register operation can be implemented through invoking the `setMessageListener()` method. The `MessageListener` acts as an asynchronous event handler for messages, and contains one method, `onMessage().`

The `onMessage` method will be called automatically whenever a new message is delivered, and all actions that need to be taken for processing the new arrival message will also need to be defined in this method.

Moreover, the `onMessage` method takes one argument of the type `Message`; this `Message` type needs to be cast to the corresponding type of the new arrival message (depending on what type of message object that new arrival message uses) in order to obtain the message body. For example, the VOEvent packet is stored in the `TextMessage` object, so we need to define the `TextMessage` object, typecasting `Message` to `TextMessage`, and then invoking the `getText()` method to obtain the VOEvent packet for further operations (i.e., displaying details of the VOEvent packet, applying some particular analysis algorithms, or using parameters in the VOEvent packet to drive the robotic telescope).

The example code 13 shows the detailed subscription processes of the "`moa.voevent`" topic, and uses MessageListener to handle incoming message from ActiveMQ.

```
//Open connection and create session

//Specifies a client name for uniquely identifies
//each durable subscription it creates
TopicSubscriber subscriber =
Session.createDurableSubscriber("moa.voevent", "clientname");

//Setting message listener for receiving messages
subscriber.setMessageListener(new MessageListener() {
   public void onMessage(Message msg) {
      TextMessage recMsg = null;
      try {
          // If the received message is type of TextMessage
          if (msg instanceof TextMessage) {
              //Typecasting
              recMsg = (TextMessage) msg;
              //Get the body of message
              String body = recMsg.getText();
              //Further operations
          }
      } catch (JMSException ex) {
          System.out.println(ex.getMessage());
      }
   }
 });
```

Code example 13: Key codes for implementing topic subscriber

We have also implemented an application called VOEventSubscriber. This application will receive all of the VOEvent packets that are published by the ActiveMQ publishing task. When the application runs, it will require end users to provide their user name and password (every end user needs to register in the ActiveMQ security domain) before they can use the application. After logging in to the application, the application will subscribe to the "`moa.voevent`" topic, and wait for a new VOEvent packet arrival.

Whenever a new VOEvent packet arrives, the application will store the VOEvent packet on the harddisk, notifying end users with sound and displaying the VOEvent packet to the end user in a human readable format. This VOEventSubscriber is typically designed for those end users that just want to receive new transient information from MOA without any extra requirements.

In fact, this application not only works for MOA and the ActiveMQ server, but it is a common VOEvent packet receiving tool, and it can be used for receiving VOEvent packets from any JMS type message oriented middleware by simply changing the server IP, port number and topic name. Figure 31 shows the full data flows of this VOEventSubscriber application.

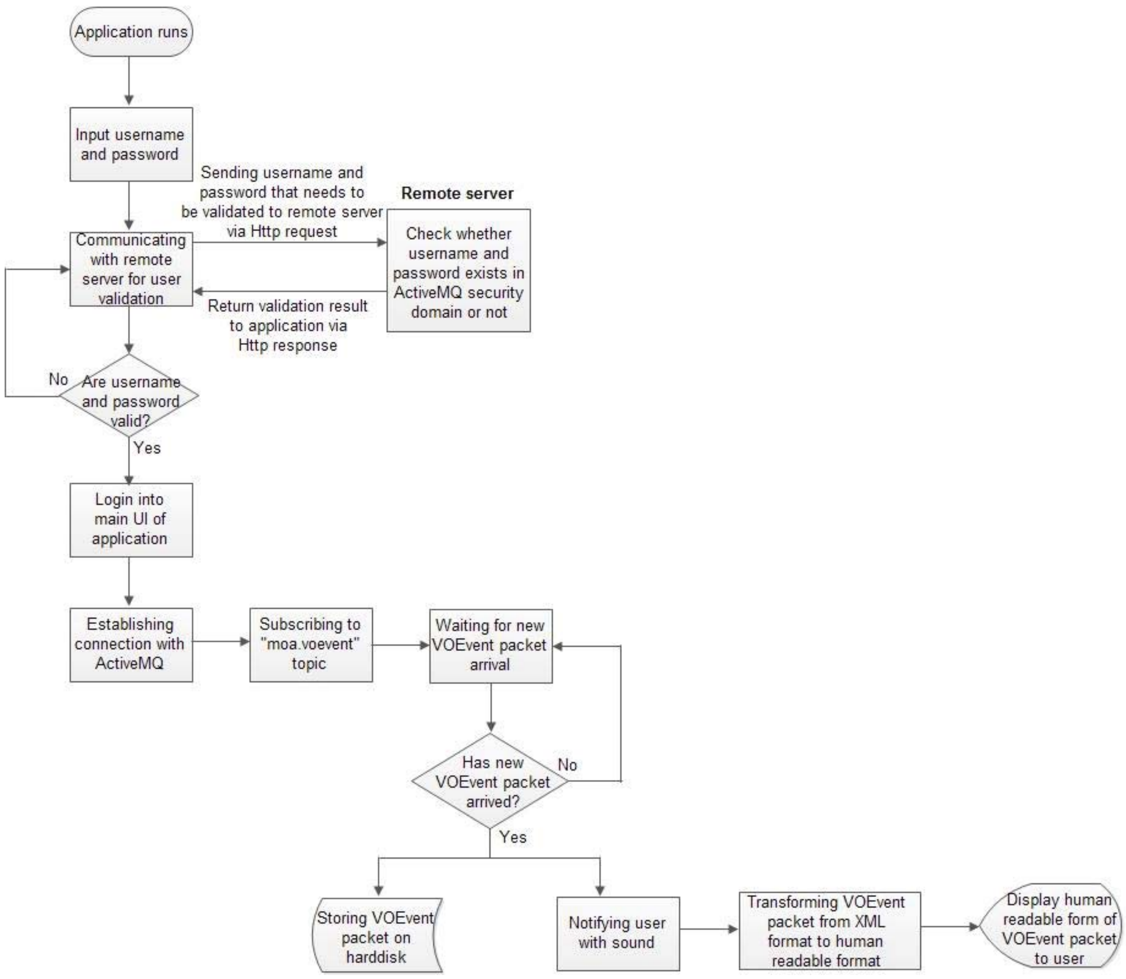

Application runs
Input username and password
Sending username and password that needs to be validated to remote server via Http request
Remote server
Communicating with remote server for user validation
Check whether username and password exists in ActiveMQ security domain or not
Return validation result to application via Http response
No
Are username and password valid?
Yes
Login into main UI of application
Establishing connection with ActiveMQ
Subscribing to "moa.voevent" topic
Waiting for new VOEvent packet arrival
Has new VOEvent packet arrived?
No
Yes
Storing VOEvent packet on harddisk
Notifying user with sound
Transforming VOEvent packet from XML format to human readable format
Display human readable form of VOEvent packet to user

Figure 31: Flow chart diagram of the VOEventSubscriber application, this flow chart diagram was produced by Edraw Max.

After the subscriber part has been implemented, the entire new transient notification architecture is complete. In the next section, we will briefly discuss how to use ActiveMQ outside the Java world.

4.6 Using the ActiveMQ Server outside the Java World

The JMS API is based on Java and it is a part of the Java platform, which means that Java must be used in order to access the ActiveMQ in a native way. However, some programmers might prefer to use other programming languages to access the ActiveMQ. Thus, ActiveMQ also supports a language agnostic protocol called STOMP. This protocol allows the ActiveMQ to be accessed by different programming languages, rather than just through the use of Java.

In this section, we will use popular script programming language Python as an example to demonstrate how to publish and receive VOEvent packets to and from the “`moa.voevent`” topic in ActiveMQ.

The Python version of STOMP API is a necessary component that needs to be downloaded in order to achieve the above goal. The STOMP official website provides a list of STOMP APIs that correspond to different programming languages. In our example, the API we use is called `stompest`, it is a full featured STOMP API typically designed for Python. This API can also be found on the STOMP official website. We can write a Python program to access the ActiveMQ via the STOMP protocol after the API has been downloaded.

Note that the STOMP protocol uses a different port to the Openwire protocol (Openwire is the default protocol used by ActiveMQ), thus, the port number must be correctly specified in the codes. Otherwise, a connection exception will be thrown when the application attempts to access ActiveMQ via the STOMP protocol. The code example 14 illustrates Python versions of VOEvent subscriber:

```
from stompest.config import StompConfig
from stompest.protocol import StompSpec
from stompest.sync import Stomp
port = '61613'
config = StompConfig('tcp://' + hostname + ':' + port)
channel = '/topic/moa.voevent'
client = Stomp(config)
client.connect(headers={'login': username,
'passcode': password, 'client-id': 'clientid'})
#defining the subscriber header
subhdrs = {StompSpec.ACK_HEADER: StompSpec.ACK_CLIENT_INDIVIDUAL}
#subscribing the "moa.voevent" topic
client.subscribe(channel, headers=subhdrs)
#waiting for new VOEvent packet arrival
while True:
  #receiving new arrival VOEvent packet arrival, the receiveFrame
  #method works in similar way as onMessage() method in Java
  frame = client.receiveFrame()
  #displaying the body of VOEvent packet
  print frame.body
  #close connection
  client.disconnect()
```

Code example 14: Python version of VOEvent subscriber

As we can see, the processes to access the ActiveMQ via the STOMP protocol are very similar to those for the default Openwire protocol. We will not discuss too much detail about the above codes, because the main purpose of this section is to demonstrate that the ActiveMQ can be very easily accessed by using other programming languages.

Another very important benefit offered by the STOMP protocol is cross language communication, which means that the subscriber application and publisher application can be implemented in completely different program languages, yet they are still able to communicate with each other. For example, we can use Python to subscribe and receive new transient event information from the "`moa.voevent`" topic, even though the publisher application is implemented in Java.

However, Python is not the only programming language supported by STOMP; STOMP supports a very wide range of programming languages (almost all modern programming languages). The example code 15 show how to use Perl to subscribe to the "`moa.voevent`" topic via the STOMP protocol:

```perl
use IO::Prompt;
use Net::STOMP::Client;

#Callback function supplied to wait_for_frames().
#This function will be invoked whenever a new frame arrives
sub doframe {
   my ($self, $frame) = @_;
   print "Got a frame: ";
   print $frame->header("message-id") . "\n";
   print $frame->body() . "\n";
   return(0);
}

#If you don't have IO::Prompt module installed,
#you can just hard code your username and password
#(but not good practice)
$username = prompt("Username > ");
$password = prompt("Password for $username > ", -e => '*');

$stomp = Net::STOMP::Client->new(
  #host IP address
  host => 'ec2-184-72-17-222.us-west-1.compute.amazonaws.com',
  port => 61613,
);

#connecting to activemq and login
$stomp->connect(
      login => $username,
      passcode => $password,
);
$stomp->subscribe(
       destination => "/topic/moa.voevent",
       id => "clientid",  # Apparently required in STOMP 1.1
       ack => "client"
);

#Now just wait for frames, invoking the given callback
#function whenever a new frame arrives
$stomp->wait_for_frames(callback => \&doframe);

#unsubscribe from topic
$stomp->unsubscribe(id => "clientid");
$stomp->disconnect();
```

Code example 15: Perl version of VOEvent subscriber

In fact, ActiveMQ is not only useful in a desktop based application; we can also access ActiveMQ in a web browser in order to develop real time web applications. To communicate with ActiveMQ in a web browser, we must first enable WebSocket transport (the procedures for enabling WebSocket transport can be found in Section 4.3.3).

The WebSocket transport implements STOMP over web socket functionalities. We can then use `stomp.js` API to access ActiveMQ via STOMP in a pure web environment. The example code 16 shows how to use Javascript to subscribe to the “`moa.voevent`” packet in web development:

```
<html>
<script src="stomp.js"></script>
<script type="text/javascript">

var client = Stomp.client("ws://ec2-184-72-17-222.us-west-
                          1.compute.amazonaws.com:61614/stomp");

//connecting to the ActiveMQ
client.connect("username", "password",
   function() { //Callback function, called after client
                //is sucessful connect to the ActiveMQ

     //subscribes "moa.voevent" topic
     client.subscribe("/topic/moa.voevent",
        function(message) { //Callback function, called when client
                            //receives new message from ActiveMQ

        alert(message.body);}) //display message content
```

Code example 16: Accessing ActiveMQ through using Javascript in web development.

We do not list all of the different programming language examples here, but all of the supported programming language API and code examples can be found on the STOMP official website. Moreover, the STOMP protocol is not only supported by ActiveMQ; we can also use STOMP API to access any JMS message oriented middleware that supports the STOMP protocol.

4.7 Performance Benchmark of Transient Notification Architecture

The new transient notification architecture is well suited to the MOA situation; it has worked for MOA over a two year period without any system faults. However, the data rate detected by MOA and the size of the VOEvent packet used by MOA are relatively small when compared to future survey projects. For example, the LSST is a future large survey. This project will include images in the VOEvent packets, and the size of the VOEvent packets might reach 1MB. Moreover, the data rate of LSST is also very high; in theory, the LSST survey will issue approximately 100,000 transient [148] notifications each observation night, which means that there will be around 3-4 VOEvent packets of 1MB size issued each second.

The main purpose of this performance benchmark is to aim to determine the scalability of this new transient notification architecture. We are using the LSST data rate as the measurement standard, because the LSST is the heaviest data rate project of all the known future transient survey projects. If our transient notification architecture has the ability to deal with the LSST data rate situation, this means that it is able to scale up and be used by other future transient survey projects for transient notification.

We will examine the performance of both the publisher part and the subscriber part. All of the benchmark tests in this chapter are conducted under the following conditions:

- The message oriented middleware used for the benchmark is ActiveMQ, the version of ActiveMQ is 5.8 during the benchmark, and no other applications were running or using resources on the machine that runs the ActiveMQ middleware.

- There are two machines involved in this performance benchmark. The machine running the ActiveMQ has hardware configuration of 2.75 GHz Intel i5-2500s CPU, 4GB of RAM, and runs a Windows 7 64bits operating system using the default installation settings. The machine running the publisher application and the subscriber application has hardware configuration of 2.75 GHz Intel i5-2500s CPU, 2GB of RAM, and runs Mac OS X version 10.7.5. The subscriber application will not run while the publisher application is running, and vice versa.

- The performance benchmark is based on the internal network and the results do not include any network latency, because if we run the benchmark under a real network environment, the different speeds of the network will give different results, so we would not be able to get accuracy in the performance results of our new transient notification architecture.

- The publisher uses persistent messaging as its delivery mode, and the subscriber subscribes to topics based on the durable subscription mode.

As previously discussed, the ActiveMQ can also be accessed using STOMP, besides the native JMS access. Thus, we have developed the JMS and the STOMP version of the benchmark applications for each performance measurement task (publisher and subscriber) in order to compare the performance between these two different access approaches.

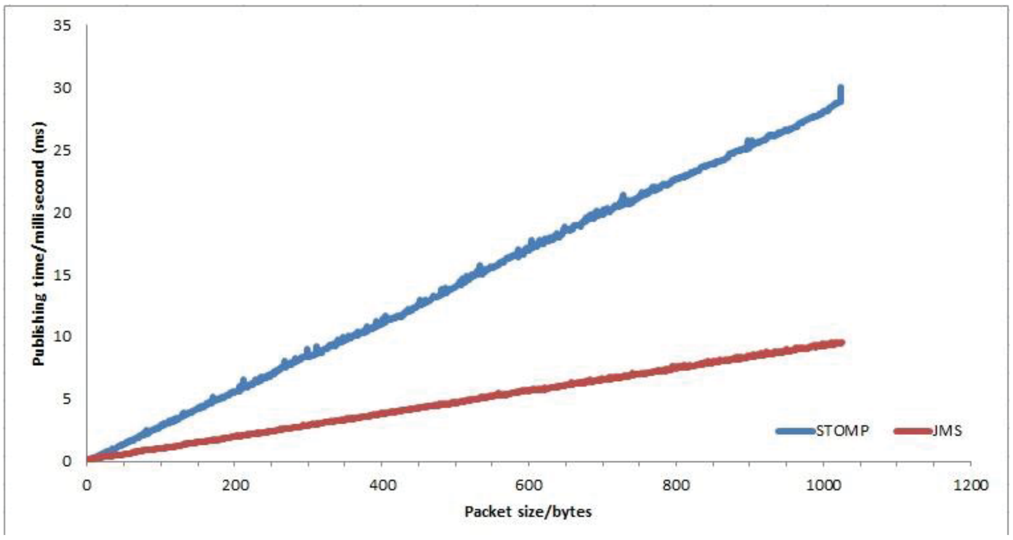
The JMS version of the benchmark application utilises the JMS API provided by the Java platform and communicates with ActiveMQ via the Openwire protocol. The STOMP version of the benchmark application is implemented based on Python, using the `stompest` API, and communicates with ActiveMQ via the STOMP protocol.

The performance data was collected by measuring the time consumed in publishing and receiving different sizes of packets to and from the topic. The packet sizes used for this measurement ranged from 1KB to 1MB.

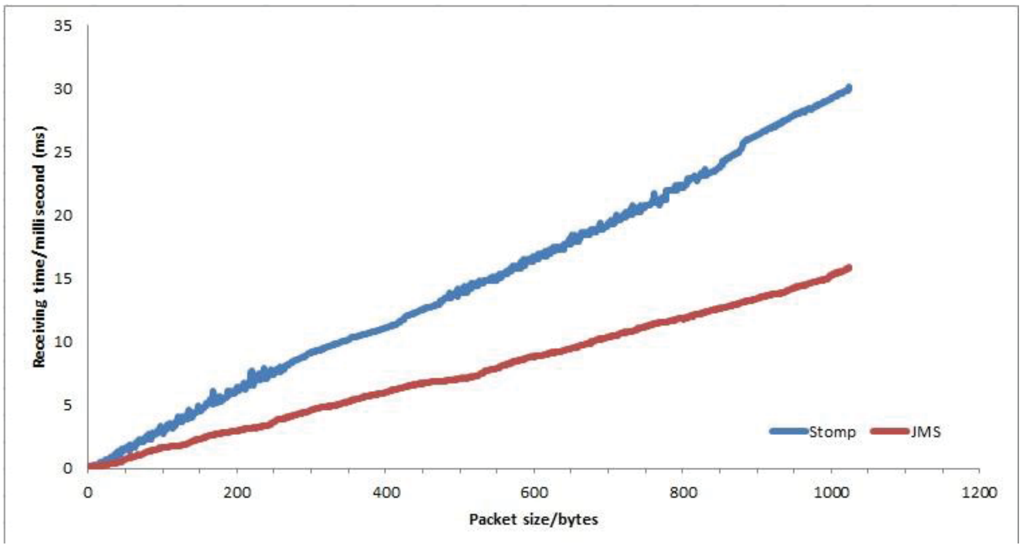
Each packet for each size was published and received 1,000 times and the total time consumed of each task (either published or received) was averaged in order to obtain accurate measurement results.

Performance measurement results and discussion

Figure 32 shows that the time consumed in publishing and receiving packets by using either JMS or STOMP increases linearly when the size of the packet is increased, because the larger packet increases the data throughput when communicating with the server.



(a) Measurement results of publishing packets to the ActiveMQ.



(b) Measurement results of receiving packets from the ActiveMQ.

Figure 32: Performance comparison between two different access approaches, with the sample of packet sizes ranging from 1KB to 1MB (X axis), and the Y axis showing the average time for processing (publishing or receiving) a single packet. The solid red line is the time consumed for processing different sizes of packets when using the JMS. The solid blue line is the time consumed in processing different sizes of packets using the STOMP.

The performance of publishing and receiving packets using native JMS access is extremely fast, but the time consumed in completing the receive packet operation is longer than the publish packet operation when using the JMS. This is because when publishing a packet, the type of packet that needs to be sent (refer to Section 4.4.4) is explicitly specified. However, when the subscriber application receives a new packet from the message oriented middleware, the subscriber application needs to determine the packet type from the header data of the received packet, and then convert the received packet to the corresponding message type before getting the actual message from the packet (refer to code example 13).These extra steps described above lead to the operation of receiving packets consuming more time for completion than does the operation of publishing packets.

However, the overall performance of JMS outperforms that of the STOMP. The time consumed in publishing and receiving packets using the JMS is nearly three times faster than when using STOMP. The main reason for this is that the ActiveMQ is built in Java on the JMS, and the STOMP is an additional protocol that is built on top of the JMS in order to enable other programming language to work with the ActiveMQ. While we send packets through using STOMP, the ActiveMQ will execute the STOMP-JMS mapping operation in order to transfer the STOMP format (‘`frame`’) to the JMS format. This mapping operation will generate an extra overhead and lead to performance decrement. Similarly, when we try to receive packets from the ActiveMQ using STOMP, the same mapping operation will also be applied, but in reverse (JMS-STOMP). Moreover, the time consumed by publishing and receiving packets using STOMP is very close, because STOMP does not need any typecasting operation as when using JMS.

Based on the measurement results, we can calculate the maximum throughput of packets per second with a size of 1MB to determine whether or not this new transient notification architecture is able to handle the LSST data rate. As we can see, for when using JMS, publishing and receiving a single packet with a size of 1MB will take approximate 9ms and 15ms, respectively. Thus, the maximum throughput will be around 100 packets/s for publishing and 60 packets/s for receiving.

On the other hand, the time consumed for publishing and receiving a single 1MB packet when using STOMP is very close, at approximately 30ms. Thus, the throughput limit of STOMP is around 33 packets/s for both publishing and receiving.

According to the results discussed above, both JMS and STOMP present impressive performance on publishing and receiving packets, and the maximum throughput of packets per second is considerably higher than the expected LSST event rate. Therefore, this new transient notification architecture is able to be upscaled and used by the LSST and other future survey projects without any problems.

Indeed, there are two important factors that will impact the performance of this new transient notification architecture. The first one is the network connection speed; if the publisher application, subscriber application and ActiveMQ server are run under a slow network environment, the maximum throughput of packets per second will be decreased. Since maximum throughput of packets will be affected by the speed of network connection. Thus, we highly recommend running the publisher application, subscriber application and ActiveMQ server under a fast network connection environment, in order to obtain better performance.

The second factor is the hardware configurations (CPU, RAMs) of the machine that runs the publisher application, the machine that runs the subscriber application, and the machine that hosts the ActiveMQ. In theory, the installation of powerful hardware means that all of the operations can be completed in a shorter amount of time. In other words, the overall performance and the maximum throughput of packets could be boosted. However, modern computers consist of complex hardware components, so the overall performance of this architecture is unpredictable when repeating the same experiment on machines with much more powerful specifications. Despite this, this new transient notification architecture could fulfil LSST requirements without any problems if the publisher application, subscriber application and ActiveMQ server are run on the same specification machines as we used in this experiment.

4.8 Conclusions

In this chapter, we have provided the detail of implementing the JMS based transient notification architecture. The structure of this transient notification architecture is very simple and consists of three major components; the VOEvent dispatcher application, the ActiveMQ, and the VOEvent receiver application. The advantages brought by this simple structure are a reduction in complexity during the system development stage and a reduction in error probability when running the entire architecture.

The JMS is also a scalable and convenient technique for delivery of transient notifications. In our system, we use the `TextMessage` format for delivery of our VOEvent packets, because the VOEvent packet is based on an XML document, so the `TextMessage` format is sufficient to deal with our requirements.

In addition, as we mentioned in Section 3.5.2, the JMS supports very rich message formats, so if future survey projects need to deliver complex data in transient notifications to end users, they can simply use other supported message formats without implementing any extra customised plugins. For example, finder chart images can be delivered through using a `ByteMessage` format.

The STOMP protocol is another remarkable technique that we have discussed in this chapter. This protocol opens the window for other programming languages to work with any JMS based message oriented middleware (message oriented middleware must support STOMP) without any limitations. All non-Java programmers can write STOMP based transient publisher and transient subscriber applications using their familiar program language, and access the JMS based message oriented middleware in the same way as when using a native JMS access approach.

The message oriented middleware we use in our architecture is ActiveMQ, but it is not a necessary component to reproduce the architecture that we propose, and it can be replaced by any other JMS based message oriented middleware that supports the STOMP protocol, such as HornetQ, OpenMQ or RabbitMQ.

However, ActiveMQ has the largest number of installations of all open source message oriented middleware, and also has the largest distribution; it is a very powerful open source message oriented middleware, and provides many highlight enterprise level features apart from those features we use in this chapter. These enterprise level features could be very useful for utilisation by future survey projects when they implement transient notification architecture.

Moreover, the measurement results indicate that ActiveMQ has an impressive performance when used in publishing and receiving large amounts of packets within a very short duration, and is suitable to handle LSST data rate. Thus, we highly recommend that those large data rate future projects could consider using ActiveMQ as message oriented middleware for transient notification.

In addition, aggregating and expanding the functionalities of the ActiveMQ based transient notification architecture is also a simply task. In the next chapter, we will introduce how to receive transient notifications from the ActiveMQ on a mobile platform.

Chapter 5 Receiving Transient Event Notifications on Mobile Platforms

In the previous chapter, we discussed in detail the use of ActiveMQ as message oriented middleware to implement the JMS based transient notification architecture, and described how to publish and receive transient notifications through using this architecture.

However, all of the applications used to receive the transient notifications described in the previous chapter are desktop based; the ability to receive transient notifications on mobile devices is an attractive idea. For some years it has been possible to receive Short Message Service (SMS) alerts of GRB events. However, in the smartphone era, a much richer user experience (easy to use, easy access to the Internet, rich user interface) is now possible.

In this chapter, we will describe how to implement a smart mobile application to receive transient notifications from the architecture that we discuss in the previous chapter, along with the details of mobile application design, and the implementation features and issues that were encountered in the development stage. (Please note, the message oriented middleware that has been used in this chapter is the same as in the last chapter, which is ActiveMQ.)

5.1 Choice of Mobile OS (Operating System) Platform

In order to receive transient notifications on a mobile device, a particular mobile OS platform needs to be selected for application implementation. A suitable mobile OS platform for this task must meet four conditions; ease of development, large user group, convenient method for distributing and installing the application, and simple to use.

There are various mobile OS platforms available in the market, such as Android, iOS, BlackBerry 10 and Windows Phone [149]. The Android operating system is a popular choice for mobile devices; it was introduced by Google Inc. [150] [151] and occupies more than 50% of the market share in the smart device operating system market [152]. Many popular mobile devices brands (such as Samsung, HTC and SONY Xperia) use Android as their device OS.

Getting into Android development is not complicated. Android uses Java as its development language, and there are a number of Integrated Developer Environments (IDEs) that provide a rich set of user interface (UI) components. Moreover, distributing Android applications is also very simple. Google provides two convenient approaches for developers to distribute their mobile applications; either submitting the APK (application package file, which is the installation file format for the Android system [153]) to the Android Application Market (similar to the App Store), or uploading the application onto a web server with a download link.

After the application has been distributed, the application users can simply install the application through the Android market or download the APK file from the developer’s website to their device’s hard drive by double clicking on the file for installation, or using a software management application to install it. There are lots of advantages offered by the Android OS; thus, the mobile application for receiving transient notification is implemented based on the Android OS.

5.2 Default Notification System of Android Application and Limitations

The popular mobile platforms come with their own push services for server initiated messaging. For example, the iOS platform for Apple mobile devices comes with the Apple Push Notification Service. An iPhone application, for astrophysical transients, that uses this service is described by Truax [154]. An iPhone app for transient alerts has also been released by the LSST collaboration [155].

The notification system used in the Android platform is called GCM (Google Cloud Messaging for Android). It was designed to replace, and overcome the drawbacks of the previous Android Cloud to Device Messaging (C2DM) notification system [156]. The GCM model is very similar to the C2DM model. It contains three main components; an application server, a GCM server, and the Android device. The application server (which can be implemented in most programming languages) is developed by the application provider, and the GCM server is provided by Google. Before using the GCM service, both the application server and the mobile device need to be registered on the GCM server.

Following registration, when new data becomes available, the application server sends a message to the GCM server via HTTP POST. The message sent from the application server is a lightweight message intended to inform the mobile application that new data are available, but without requiring transfer of all the data to the application. When the GCM server receives the message, it will immediately route the message to the mobile device if the device is online. Otherwise, the message will be held and delivered once the mobile device is back online [156].

Once the message is successfully delivered to the mobile device, the mobile application will start up, process the message, and fetch the new data from the application providers' server. Moreover, the application should continue running (either in the foreground or background) in order to receive messages from the GCM server. When the application is not running, it will not be able to receive any new messages at the GCM server.

The GCM service provides the ability for application users to receive new event notifications rapidly. However, there are some issues with the GCM service as described below.

1. Enabling the GCM service in the Android platform is a complex task, requiring the developer to spend considerable effort in coding the application server and the Android application.

2. The GCM service has two OS version restrictions. The first restriction is that only those devices with Android 2.2, or higher, are able to use the GCM service. In other words, if the device owner does not upgrade their Android OS to at least version 2.2, they are not able to receive all the functionalities provided by the GCM service. The second restriction is that those devices with Android versions higher than 2.2, but lower than 4.04, are required to set up a Google account and install the Google Market on their device in order to use the GCM service. As of writing this thesis, the current version of Android OS is 4.4. If users update their device to the newest OS version, they would be able to use the GCM service.

3. The GCM has limits on the payload of each message. It only allows 4 KB of pure text message to be sent to the device. In some application contexts, however, the application provider may wish to send other types of data (such as imaging and/or graphical data), or messages of more than 4 KB directly to the application. As mentioned above, the provider cannot send messages directly to the application. Every message is required to pass through the GCM server. If imaging and/or graphical data are included, the message could easily end up being more than 4 KB and thus be rejected by the GCM server.

4. The GCM has rate limits for notification messages on each application on each individual device. These limits were designed to prevent malicious application providers sending huge amounts of spam messages to the device. When the rate limits are exceeded, the notification feature of the application will be suspended for a few minutes. The application is not able to receive any instant notifications during the suspension period. This limitation will have an impact on the transient event application, because new generation surveys (such as LSST) are expected to detect huge amounts of transient events in a short period. Therefore, even though

new generation surveys employ a filtering feature that allows the user to select the event types they want to receive in the mobile application, the notification message rate limits might still be exceeded.

These limitations have prompted us to consider using message oriented middleware as an alternative approach for delivery of our transient notifications to Android device. With the message oriented middleware, we can avoid all issues encountered in the GCM. The MOM broker will not block the communication channel in the case of high frequency messaging, and any version of Android can communicate with the MOM broker through the application program interface (API). In the next section, we will explain how to communicate with the ActiveMQ broker in the Android application.

5.3 Connecting the ActiveMQ in the Android Application

There are two approaches that can be used for communicating with the ActiveMQ server from an Android application. One approach is to use a RESTful (Representational State Transfer) API, provided by ActiveMQ. A RESTful API provides the ability for the mobile application to connect and retrieve instant messages from ActiveMQ via the HTTP GET method. However, with this approach, the mobile application is only able to make use of the basic features (connecting to the server, temporarily subscribing to the topic, retrieving the new data) offered by ActiveMQ. In our mobile application we wish to utilise the advanced features (durable subscription, sending and receiving persistent messaging, keeping the mechanism alive) provided by ActiveMQ. This limitation is in the implementation of the REST API that is provided by the ActiveMQ server. The REST API only defines the basic features as described above for accessing ActiveMQ.

The other approach is to use the STOMP technique discussed in the previous chapter. With STOMP, our Android application is able to access the full functionality and advanced features provided by ActiveMQ (durable subscription, sending and receiving persistent messaging, keeping the mechanism alive).

To enable the Android application to communicate with the ActiveMQ server via STOMP, it is necessary to implement support for STOMP on the Android device. As we all know, the Android application is implemented in the Java programming language, and Java support for STOMP is included in the ActiveMQ library. However, if we try to directly import the ActiveMQ library file in the source code for an Android application, a compilation exception will be thrown up by the Android compiler. The reason for this is that the Android run time environment is quite different from that implemented on desktop and server PCs. The Android library framework is, therefore, not equivalent to the Java Platform Standard Edition (SE). This is not an issue with any particular IDE.

The source code of the ActiveMQ library uses some core libraries that only belong to the Java SE, and these are not supported by Android. Thus, we need to strip out the unnecessary code from the ActiveMQ library, leaving us with a smaller library that specifically supports STOMP. The resulting Java bytecode cannot be run directly on Android because Android applications run on the Dalvik virtual machine rather than the standard Java Virtual Machine (JVM). To produce code that can run on Android, the dx tool is required in order to transform JVM bytecode to Dalvik bytecode.

The following instructions show details of how to strip out the STOMP support from ActiveMQ and convert it into bytecode for Android devices.

1. Download the ActiveMQ distribution file from the ActiveMQ official site (both Windows version and Linux version distribution files are available on the site; the user needs to select the one that corresponds to their operating system).

2. Unzip the distribution file and switch to the root directory of the distribution file; you will see a jar file called activemqall-x.x.x. Where x.x.x is the activemq version code; in our example it is activemqall-5.8.0.

3. Create a new directory on the Desktop, and open the activemqall-x.x.x jar file with an appropriate application (e.g. 7-Zip). Copy all sources within the jar file into the new directory.

4. The new directory should contain three sub-directories (javax, META-INF, org) and three files (activemq.xsd, activemq.xsd. html and activemq.xsd.wiki). Delete all the subdirectories shown in Table 5:

Directory to delete	Path to directory
`javax`	Root of jar file
`fusesource`, `jasypt`, `slf4j`	`org`
`log4j`	`org/apache`
`advisory`, `blob`, `broker`, `camel`, `command`, `console`, `filter`, `hooks`, `jndi`, `kaha`, `leveldb`, `management`, `memory`, `network`, `openwire`, `plugin`, `pool`, `protobuf`, `proxy`, `security`, `selector`, `spring`, `state`, `store`, `transaction`, `usage`, and all `.class` files	`org/apache/activemq`
`filter`	`org/apache/console`
`discovery`, `failover`, `fanout`, `http`, `https`, `logwriters`, `mock`, `mqtt`, `multicast`, `nio`, `peer`, `reliable`, `tcp`, `udp`, `util`, `vm`, `ws`, `wss`, and all `.class` files	`org/apache/transport`

Table 5: List of sub directories needing to be deleted.

5. Compile the source files of STOMP and place the compiled file into JAR format.

6. Use the `dx` tool as shown below to convert the Java bytecode to Dalvik bytecode. The `dx` command tool is provided by the Android SDK, and is located in the

platform-tools directory under the Android SDK root directory. After the jar file has been converted to Dalvik bytecode, it is ready for use in Android development.

```
dx -dex -keep_classes -output=input.jar output.jar.
```

7. The last step is to create an Android project using a suitable IDE. For example, the Eclipse IDE is available from: http://www.eclipse.org/downloads/. Copy the Dalvik bytecode jar file into the lib folder under your Android project folder.

Once the new library has been imported into the Android project, we can develop the code for communicating with the ActiveMQ via the STOMP protocol in the Android application, and the Android application will be able to access the full STOMP functionality provided by the STOMP library.

5.4 Design of the Android Application for Receiving Alerts of Transient Events

Designing a mobile application is different from designing a native desktop application. The mobile application design should focus on simplicity, ease of use and durability aspects with minimum usage instruction being required, because the main idea of a mobile application is to provide the ability for users to explore information rapidly. Thus, the mobile application should not supply too many options and it should also present information in a succinct manner. The Android SDK (Software Development Kit) provides a rich set of UI components that allows developers to utilise any desired functionality provided by the Android platform. Moreover, once the application has been developed, it can be installed in all mobile devices that support the Android OS.

Our application for receiving notifications of MOA events was developed with the following design goals in mind:

1. Alerting the user when a new transient event has been detected;
2. Displaying a list of previously detected events for further examination;
3. Labelling the events that have been read;
4. Displaying the summary details of specified events with a finder chart;
5. Storing all data from previously received events' data on the storage space of the device (this will be a solid-state drive on a modern smartphone);
6. Removing specified events from storage; and
7. Redirecting to the MOA web pages to view the full event profiles after the user clicks the hyperlink in the specify event.

Our transient application provides a fast method for users to receive and read the summary details of the newly detected events in their mobile devices wherever they might be at the time that the event is detected.

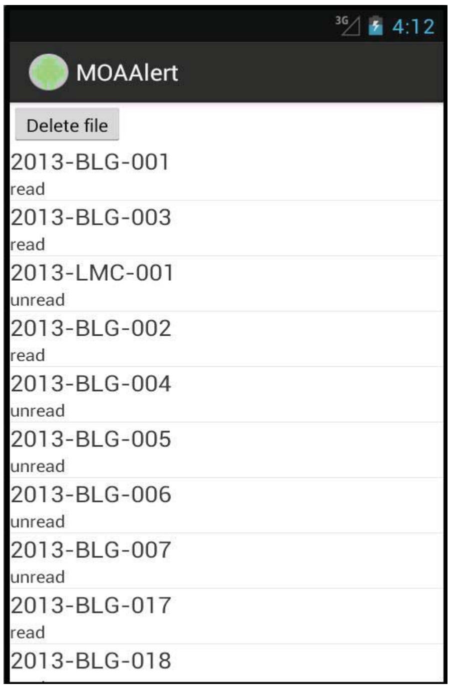
5.4.1 Features of the User Interface

Once the user launches the application, the application will list all the recently received events, as shown in the screenshot in Figure 32 (a). This list presents the name of each event with a label underneath the event name. The label feature is useful when users have a large number of event files kept on their device, because this feature will let the user know whether or not the specified event has been read. The event list displays the ten most recent events on the screen at one time, but more events may be viewed by scrolling the list.

If the user wishes to view the details of a particular event, they can simply tap the event name on the list. The application will load the corresponding VOEvent file from storage, parse the event file from VOEvent form to human readable form, and then display the important parameters of the event to the user. A screenshot of a displayed event is shown in Figure 33 (b).

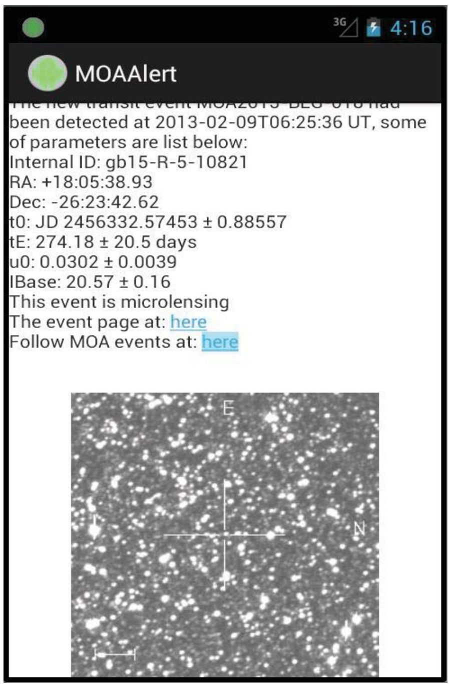
Also displayed is a finder chart for the event. This chart is actually stored in a remote server as a JPEG file. The URL of this file is included in the VOEvent description and the file is fetched each time the display feature for individual events is invoked. This is because the available storage space in mobile devices is small and storing finder charts for each event could overrun the capacity. Note that fetching a file from the remote server requires network interaction and it could be a long running operation due to a slow network connection environment. For this reason, it is not good practice to perform this operation in the UI thread, because if the fetch image file operation is performed in the UI thread, the user interface of the application will "freeze" until the fetch operation is completed, which will lead to a bad user experience.

Therefore, we need to utilise the AsyncTask class [263] and perform the fetch image operation within this class. The AsyncTask class allows for the performing of operations in the background and update results on the UI thread after operations are completed. In other words, upon the user clicking an event name, as shown in Figure 33 (a), an instance of the AsyncTask class is created and the fetch image operation is performed in the background. In the meantime, the user interface will remain active and the user will still be able to use the application. The image will be displayed on the screen (see Figure 33 (b)) after it has been completely fetched from the remote server. Invoking the fetch image operation in the AsyncTask class achieves the best utilisation of the processing power and memory of the device.



2013-BLG-001
read
2013-BLG-003
read
2013-LMC-001
unread
2013-BLG-002
read
2013-BLG-004
unread
2013-BLG-005
unread
2013-BLG-006
unread
2013-BLG-007
unread
2013-BLG-017
read
2013-BLG-018

(a)

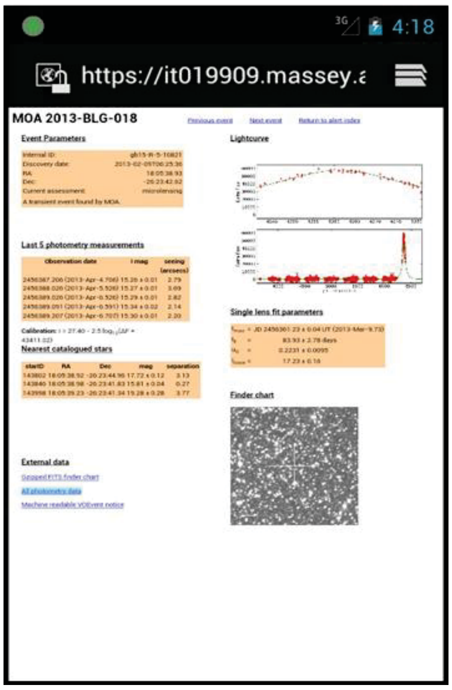


been detected at 2013-02-09T06:25:36 UT, some
of parameters are list below:
Internal ID: gb15-R-5-10821
RA: +18:05:38.93
Dec: -26:23:42.62
t0: JD 2456332.57453 ± 0.88557
tE: 274.18 ± 20.5 days
u0: 0.0302 ± 0.0039
IBase: 20.57 ± 0.16
This event is microlensing
The event page at: here
Follow MOA events at: here

(b)

Figure 33: The left hand side figure showing a list of events received by the Android application, for each event a status message specifies whether or not the event information has been read by the user. The right hand side figure is a screenshot of a display of the details of a selected event. Here the displayed parameters are; the event name, the date the event was detected, the equatorial coordinates, the details of the best fitting microlensing model, and a finder chart image.

The user is also able to access a full description of the event on the MOA alert website, by tapping the hyperlink in the event display on the device. When the user taps the hyperlink, the application will trigger the default device browser and display the web page in the browser, as shown in Figure 34 (a). Finally, the application provides a user interface to remove unwanted event files from the device storage, as shown in Figure 34 (b).


4:18

https://it019909.massey.a

MOA 2013-BLG-018 Previous event Next event Return to alert index

Event Parameters

Internal ID:	gb15-R-5-10821
Discovery date:	2013-02-09T06:25:36
RA:	18:05:38.93
Dec:	-26:23:42.02
Current assessment:	microlensing
A transient event found by MOA.	

Last 5 photometry measurements

Observation date	I mag	seeing (arcsecs)
2456387.206 (2013-Apr-4.706)	15.26 ± 0.01	2.79
2456388.026 (2013-Apr-5.526)	15.27 ± 0.01	3.69
2456389.026 (2013-Apr-6.526)	15.29 ± 0.01	2.82
2456389.091 (2013-Apr-6.591)	15.34 ± 0.02	2.14
2456389.207 (2013-Apr-6.707)	15.30 ± 0.01	2.20

Calibration: I = 27.40 - 2.5 log$_{10}$(ΔF + 43411.02)

Nearest catalogued stars

starID	RA	Dec	mag	separation
143802	18:05:38.52	-26:23:44.96	17.72 ± 0.12	3.13
143840	18:05:38.98	-26:23:41.83	15.81 ± 0.04	0.27
143998	18:05:39.23	-26:23:41.34	15.28 ± 0.28	3.77

External data

Gzipped FITS finder chart

All photometry data

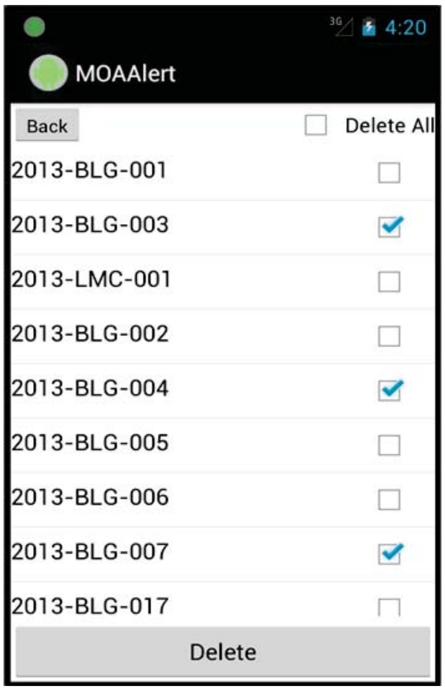
Machine readable VOEvent notice

Lightcurve

Single lens fit parameters

t_{max}	=	JD 2456361.23 ± 0.04 UT (2013-Mar-9.73)
t_E	=	83.93 ± 2.78 days
u_0	=	0.2231 ± 0.0095
I_{base}	=	17.23 ± 0.18

Finder chart


4:20

MOAAlert

Back	☐ Delete All
2013-BLG-001	☐
2013-BLG-003	✓
2013-LMC-001	☐
2013-BLG-002	☐
2013-BLG-004	✓
2013-BLG-005	☐
2013-BLG-006	☐
2013-BLG-007	✓
2013-BLG-017	☐

(a) (b)

Figure 34: The left hand side figure is a screenshot showing the web page for a particular MOA event, as displayed on the Android device. The right hand side figure shows the interface for deleting stored events from the device.

5.4.2 Handling Notifications

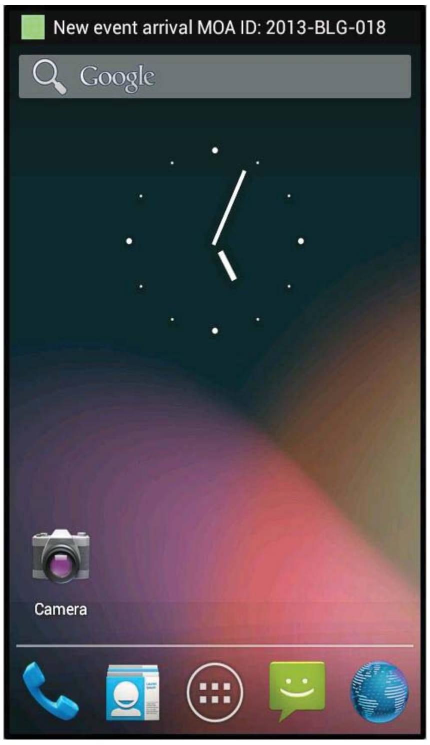
The key code components of this Android application are those that connect with ActiveMQ via the STOMP protocol, subscribe to the "`moa.voevent`" topic and receive new transient notifications from the ActiveMQ. However, if we try to write these codes in a normal Android class file, when the user switches to another application after running the transient event application, Android may kill the application. The user will then not be able to receive any new event data until the next time they launch the application. The cause of this problem is that the Android system only allows one application at a time to run in the foreground. All other applications will be run in the background and the Android system will decide for itself when it needs to close them. Any background application could be terminated by the Android system, especially if there is a low level of system memory available, or when the application is inactive for a long time.

In order to resolve this, we use the ‘‘`Service`’’ component of Android to deal with the network communication and receive transient notifications. This component is typically designed to perform long running operations in the background, where all contained routines continue to be executed until the `stopService()` method is invoked. The `stopService()` method will be called by the application if the user forcibly shuts it down, or where some unexpected error occurs in the running of the application. The Service component also provides a restart feature, which means that if the application is killed by the system, it will restart automatically without user interaction (Note: The Service component has high priority in the Android system, and it will only be killed when the system’s memory is very low). With the Service component, users are still able to receive new event alerts even when the application is being run in the background.

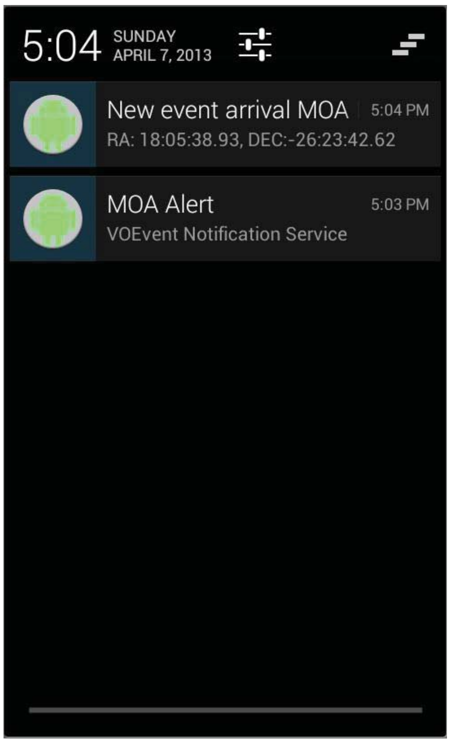
When the application receives new event data (as a VOEvent XML packet) from the ActiveMQ server, it will store the new event data as an XML file in the device storage and use the Notification feature provided in the Android API to create a notification message to alert the user.

The application has two notification modes for alerting the user in different application usage situations. One mode is invoked if the user uses the event list interface, as shown in Figure 33 (a). Here the application will vibrate the mobile device and display the name of the new event at the top of the list. The other notification mode is invoked if the user is performing other application tasks (for example, viewing other events’ details, or trying to remove event files from the device storage), or running the transient event application in background mode. If the user switches the mobile device to the sleep mode, the application will apply the second notification strategy to notify the user when a new event arrives at the application.

When a new event arrives, the application will vibrate the mobile device and display the notification message on the ‘‘notification bar’’ of the Android system, along with an alert sound. The notification bar is a feature of the UI, provided by the Android OS that displays notification messages from all applications on the device. Figure 35 shows an example of the notification system of the transient event application. All new event arrivals are processed concurrently in their own threads. An event thread will read the incoming VOEvent packet and store it as a file on the device storage.


New event arrival MOA ID: 2013-BLG-018

(a)



New event arrival MOA 5:04 PM
RA: 18:05:38.93, DEC:-26:23:42.62
MOA Alert 5:03 PM
VOEvent Notification Service

(b)

Figure 35: Screenshots of the two display modes for new event notifications.

The process of writing data to storage is treated as a critical section by the OS and is carried out one thread at a time. However, since each event is processed in its own thread, an event will not be missed if it arrives during the processing time of another. If many events arrive in a short period of time in a ‘‘burst’’ pattern, the application will notify each single event to the user, but only the last five events will be displayed on the notification bar.

The application also allows users to view the new event details by clicking the notification message on the notification bar. Upon tapping the notification message, if the application is being run in background mode, the Android system will switch the application to foreground mode and present the new event details in the event detail interface. If the user is performing another application task, then when the user taps the notification message, Android will switch directly to the event detail interface. Figure 36 shows the internal work flow of the notification system.

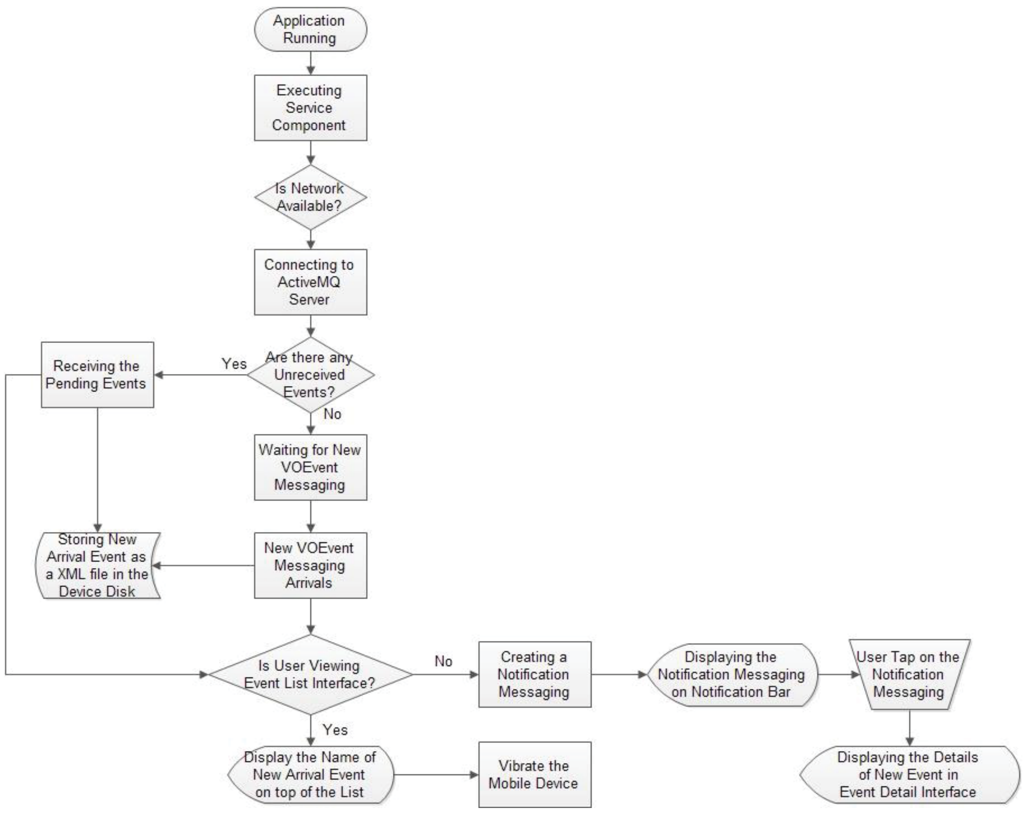

Application Running
Executing Service Component
Is Network Available?
Connecting to ActiveMQ Server
Are there any Unreceived Events?
Yes
No
Receiving the Pending Events
Waiting for New VOEvent Messaging
Storing New Arrival Event as a XML file in the Device Disk
New VOEvent Messaging Arrivals
Is User Viewing Event List Interface?
No
Yes
Creating a Notification Messaging
Displaying the Notification Messaging on Notification Bar
User Tap on the Notification Messaging
Display the Name of New Arrival Event on top of the List
Vibrate the Mobile Device
Displaying the Details of New Event in Event Detail Interface

Figure 36: Internal work flow of the Android application for receiving transient events, this flow chart diagram is produces by Edraw Max.

5.4.3 Network Issues

When the application successfully establishes the connection with the ActiveMQ server, the socket connection between the application and the ActiveMQ server will be held (also referred to as the long connection technique) until the disconnect function has been called. However, the connection could be interrupted unexpectedly during runtime. A typical example is losing a connection between the application and the ActiveMQ through making a phone call. In this case, the Android operating system will disable the network feature of the mobile device until the phone call is completed.

Thus, the transient event application must provide a feature for automatically re-establishing the connection with the ActiveMQ if the connection is interrupted. The automatic reconnect feature can be implemented through a try and catch block, where the try block calls a method for communicating with the ActiveMQ server (connects with the server, subscribes to the topic and retrieves the new messages). Once a network exception has occurred, the catch block will determine the status of the mobile device network before attempting to re-establish the connection with the ActiveMQ server. The code for determining the network status will execute every 30 seconds if the device network is not available. Once the device network is available, the catch block will call the same method as in the try block to re-establish the connection pipeline with ActiveMQ. The following pseudo-code illustrates this procedure.

```
try
  Call Communicating Method
catch
  Check Network Status
  if Network Is Not Available
     Wait For 30 Seconds
     Check Network Status
  else
  Call Communicating Method
```

Code example 17: Pseudo-code for handling network exception

Moreover, when the mobile device reconnects with the ActiveMQ server, it needs to re-subscribe to the channel, and then check whether there any un-received events pending on the server. If there are, the application will then receive all pending events from the server. The re-subscription operation happens at the time of initialising the connection between the mobile device and the ActiveMQ server.

From a usability point of view, a well-designed transient event application should still be usable without a network connection. Sometimes the user may wish to check the details of recent events in the absence of mobile network or WiFi support, or only a very weak mobile signal. In order to ensure that our application is usable all the time, we have decided to store all received events in the device storage. This will ensure that the user is still able to use the application to read all of the recent events' details even when the network connection is unavailable. We also provide a feature that allows the user to remove unwanted event files from the device for space saving purposes.

5.5 Performance Benchmark and Scalabilities Discussion

The performance benchmark of this Android application is mainly focused on measuring the time consumed in receiving and processing packets (from received packets from ActiveMQ via STOMP to displaying notifications) on the Android application. Meanwhile, we run this Android application on a real device for the purpose of performance measurement. The entire performance experiment is conducted under the following conditions:

- The device model that is used for the installation and running of the Android application is a Samsung Galaxy S3; this device has hardware configuration with 1.4 GHz Samsung Exynos 4412 CPU, 1GB of RAM and runs Android OS 4.0. Moreover, there were no other applications running (except system applications) during the performance benchmark;

- The application will access the remote ActiveMQ server via a 3G mobile network. In theory, the 3G mobile network will provide peak data transfer rates of at least 200 Kbit/s. Thus, we assume the network speed in this benchmark is 200 Kbit/s; and

- The sample size of packets in this benchmark is the same as used for the performance benchmark in Chapter 4, ranging from 1KB to 1MB. However, as we know, a mobile device is different from a desktop computer; it only has a limited amount of storage space, and mobile networks also have monthly network bandwidth usage limits. This means that if we try to receive each size of packet 1,000 times on the mobile device as what we did in Chapter 4, the total amount of 1.5 GB network bandwidth will be consumed and the bandwidth usage will definitely be over our usage limit. Thus, we reduce the receiving frequency of each size of packet from 1,000 times to 500 times in this benchmark.

The measurement results of our Android application were present in Figure 37 below:

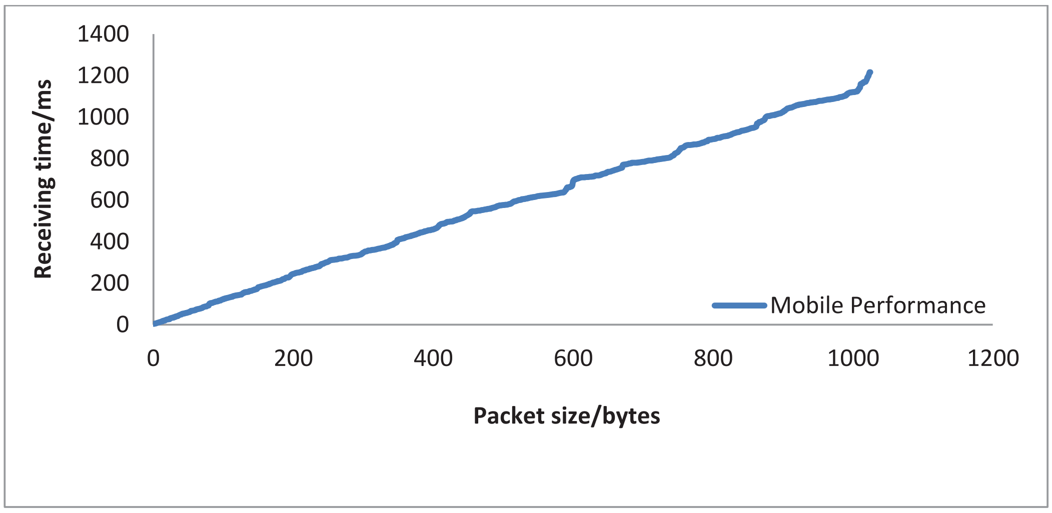


Figure 37: Measurement results of receiving different sized packets under the 3G environment with a Samsung Galaxy S3 device; the time consumption of receiving each size of packet is shown as the solid blue line in the above plot.

As we can see, the linear increase of receiving time with packet size on the Android application is very similar to the desktop based application. However, the average time consumed for receiving and processing packets on the Android application is slower than the desktop based application.

Even though it is slower than the desktop based application, the performance measurement results of the Android application are still acceptable and will satisfy most projects' requirements. However, this performance result is not sufficient to meet LSST expectations, because the measurement results show that receiving and processing a single packet with a size of 1MB takes more than 1 second for completion.

Indeed, the overall performance of this Android application is impacted by two main factors. First, the network environment is very important for Android application, because the receiving packets operation requires data transfer between the Android application and the ActiveMQ server. In other words, the time consumed in receiving packets is dependent on the mobile net connection speed. For example: if the Android application is communicating with the ActiveMQ server under a fast network environment (i.e. Wi-Fi with 1 MB/s), the data transfer speed will be much faster than for the 3G network environment. Fast data transfer speed means less time is required to complete the receive packets operation from the ActiveMQ server.

Second, after the Android application has received packets from the ActiveMQ server, it will perform an I/O operation in order to store the VOEvent packets on the device storage. That the storage operation depends on the CPU and RAM is common knowledge; a slow CPU and small RAM capacity means that more time is required to complete the storage operation. As we can see, the hardware configuration of the experimental mobile device is much lower than the experimental machines used in Chapter 4, so the Android application will take a longer time to store VOEvent packets into the storage unit than the desktop based application.

In fact, some of the most recent smart mobile devices have powerful hardware configurations, and the 4G mobile network has the ability to offer peak data transfer rates of approximately 100 Mbit/s. If this benchmark is performed on these modern smart mobile devices and the 4G mobile network environment, the performance measurement results will be at least 3 times faster than the measurement results presented here. Moreover, according to Moore's law [157], the CPU performance and RAM capacity will double every 18 months. When the LSST comes online in 2020, mobile devices will be very powerful, and the 5G mobile network [158] should be rolled out in the same year as well. Thus, this Android application should have no difficulty scaling up for the LSST and handling the LSST data rate.

In addition, as discussed above, our mobile application stores the VOEvent packet for every event it receives on the device storage. A high end smartphone would typically have around 20 GB of free storage space after the OS and all core applications have been installed. This is easily enough for our current purposes given the low event rate and small size of the packets (the MOA VOEvent packet is less than 5 KB).

However, the LSST will generate huge amounts of data and the packet size will go up to 1MB. If we apply the same policy to the LSST situation, the device storage will be filled in a very short timeframe. Apart from this device storage issue, the device battery is another issue to be considered. As network communication is one of most battery consuming operations in mobile devices, if our application keeps receiving packets at high frequency, the battery will run flat very quickly, and end users would not like their mobile device to have a battery flat within one or two hours.

Thus, this policy clearly needs to be revised to face LSST type data rates. In this situation a filter could be employed to select those events that are to be stored (perhaps implemented as a custom XPATH or XQUERY string). Filtering could be implemented at either the server side, or the application side - or possibly both. If filtering is implemented on the Android device, then the application will still receive every packet from the server and need to parse the VOEvent packet for each event. This operation was estimated to take ~100 ms in our application tested on the experimental device. Thus, this application would definitely struggle and the battery issue will still exist if directly exposed to the LSST ‘‘firehose’’. This would be an even more serious issue if graphics and imaging data (finder charts, lightcurves etc.) were also packaged in the VOEvent.

On the other hand, a server side filter will reduce the network traffic because the server will only push those events that match the users’ filter criteria. A custom topic for each user could be established on the MOM broker, which is subscribed to when running the Android application. We note that SkyAlert provides the capability for users to make custom feeds. Thus, the server side filter can fully resolve storage and battery issues, and it is the best option for LSST type data rates.

5.6 Conclusions

In this chapter, we view the use of an MOM broker (ActiveMQ) as an attractive alternative to using the GCM model native to Android. We have developed an Android application to receive notifications of astrophysical transient events, where the use of an open application layer protocol for application-server communication can, in principle, allow a variety of heterogeneous applications to subscribe to event notifications.

Developing a mobile application for transient events has the aim of increasing interaction between the survey and follow-up projects. The mobile application enables astronomers to be instantly alerted to newly discovered event data through the notification feature. The application also provides astronomers with the ability to quickly view and access the profiles of transient event data on their devices in a convenient manner. The application that we propose could be scaled up to meet the requirements of other surveys. Other projects do not necessarily have to deploy an ActiveMQ broker to participate. The important point is that they utilise a broker that supports STOMP, either by deploying their own server, or by making use of a server provided as a public hosting service.

The functionality and UI of our Android application was designed around the specifics of the metadata of events from the MOA Project. However, this could be adapted to a more general application that can handle VOEvent packets from a variety of sources of astrophysics transients.

Chapter 6
NoSQL Database for Astronomical Data Management

Modern astronomy projects, including transient surveys, generate huge amounts of different types of observational data. Such data includes photometry measurements, star catalogues, transient event coordinates, and so forth. All of these data will be stored on the disks for analysis, modelling and other purposes. (i.e. for generating VOEvent packets for transient notifications). Thus, data management is another important topic in the astrophysics transient research field. A good data management approach will provide support for managing transient data in an efficient manner, high performance access to large datasets, easy data maintenance, and easy up-scales to meet future requirements.

In the current stage, all of the existing surveys use two major approaches for their data management; either storing data on a file based system with a flat file format, or storing data in a relational database management system (RDBMS). Indeed, RDBMS is a very popular data management approach and many existing astronomical databases are implemented based on RDBMS, such as the Sloan Digital Sky Survey (SDSS) [187] and WFCAM [188].

However, these two data management approaches might facing scalability issues when vast quantities of data are needed to be stored and processed. The NOSQL database system is another type of data management approach; it offers strong scalability and supports flexible schema for data management, this type of database is typically designed to deal with a ‘big data’ situation. In fact, lots of large Internet enterprises are adopting a NOSQL database as their data management approach, and moving parts of their old data from a RDBMS to a NOSQL database. These enterprises include Google, Facebook, Amazon and LinkedIn.

In this chapter, we will explore the NOSQL database system and discuss how to use a NOSQL database to store and manage astronomical transient data.

6.1 Related Work

The database system is a very important tool for data management, because it offers the ability to manage, access and query data in an efficient manner. Many existing studies have discussed the use of relational databases for astronomical data management. Gray et al. [277] presented the design of a database system called SkyServer, which allows public astronomers and scientists to access data released by the Sloan Digital Sky Survey (SDSS) [187] through a browser. They use a Microsoft SQL server [278] for the database implementation for archiving SDSS data. In this paper, they describe their database design in detail, including the relational schema design, the tables designed for different observation data, and the design of the data access features (views, indices, and user defined access functions). In addition, they also list 20 different SQL query examples for lookup of different types of observational data from Skyserver database system. They conclude that using views and many indices give convenient access to the conventional data, and that the performance of data lookups can be improved through defined store procedures and indices.

Tankar et al. [279] proposed a Catalog Archive Server (CAS) model for accessing catalog data produced by SDSS. This model contains a set of RDBMS user defined functions and it is implemented on top of the RDBMS system. As they conclude, the use of these user defined functions could enhance the performance of accessing particular pieces of catalog data from the relational database.

Power [280] conducted a performance study between MySQL [281] and Oracle [282] databases, using SQL queries for lookup of particular star data (i.e: Searches of white dwarf stars, lookup of objects that are close to each other, and searches of objects with *rmag* values less than 21.75, among others) from astronomy catalogues that contain approximately a billion objects. Based on his results, most of the queries can be performed very quickly, and the query response time of the Oracle database outperforms that of the MySQL database. However, he also points out that the query response time of searching neighbor stars for a catalogue of a billion objects and cross matching two catalogues consisting of a billion objects is very slow, and that these two types of searches are not well supported for large catalogues in a database when only using SQL.

Many similar studies have also been carried out in the astronomical domain to investigate the feasibility of using relational database systems for astronomical data management [283] [284] [285]. However, the studies into using relational databases for data management do not only exist in terms of the astronomical domain; there are also many studies about relational database systems that have been conducted in other domains for discussing the schema design of relational database and performance evaluation, among others [286][287][288].

However, as data grows very quickly, the relational database faces a scalability issue; because most of the current systems will store and process vast amounts of data, processing these vast amounts of data requires speed, flexible schema and high scalability, which have pushed relational databases to their limits [289]. Mohamed et al. [275] state that the main scalability limitation of relational databases is that they depend on vertical scalability, which means that the only way to improve the capacity of a relational database system is by adding more powerful hardware resources on the machine that hosts the relational database system. However, each machine has a hardware limit and, once that hardware limit is reached, there is no way to improve the capacity of the relational database system. They also discuss the fact that horizontal scalability is an alternative solution to overcome this issue with vertical scalability, but that relational databases do not provide good supports for horizontal scalability.

Since there are some limitations of relational databases in terms of storing and processing vast amounts of data, another type of database has become popular; namely the NOSQL database. The NOSQL database is typically designed for dealing with big data situations and uses the horizontal scaling approach to boost database capacity [275].

The origin of the NOSQL database can be related to the Bigtable [290], which was developed by Google. The Bigtable is a distributed data storage system and Google uses it to manage their projects' data, including web indexing, Google Earth and Google Finance. The current user groups of NOSQL databases are increasing rapidly, because NOSQL databases offer high scalability and have the ability to efficiently mine data from vast amounts of datasets. Therefore, many studies have been conducted to investigate this type of database.

Couchbase [291] conducted a survey to investigate the reasons that drive people to migrate from relational databases to NOSQL databases; 49% of the survey respondents cited the rigid schema of RDBMS as the primary reason they decided to migrate from RDBMS to NOSQL, but the inability to scale out data (35%) and the high latency/low performance of RDBMS are also high ranking reasons.

Li and Monoharan [292] carry out a performance comparison study to compare MS-SQL with several NOSQL database products, including RavenDB, MongoDB, CouchDB, Cassandra, Hypertable and Couchbase. They examine the performance of read, write and delete operations between these different database products through using 6 different testing samples (10, 50, 100, 1000, 10000, and 100000). According to their findings, not all the NOSQL databases perform better than the MS-SQL database; the CouchDB and RavenDB show bad performance on all three testing operations. Cassandra shows good performance on the write and delete operations, but does not perform well on the read operation. The Couchbase and MongoDB are the fastest two database products for the read, write and delete operations in all comparison products, and these two databases show better performance than MS-SQL. However, what type of data and queries they used in their performance measurement has not been reported in the paper.

Nyati et al. [293] also conducted a study to compare read and write performance between RDBMS and NOSQL databases. In this study, they use MySQL as the RDBMS implementation and MongoDB as the NOSQL implementation. Based on their evaluation results, the MongoDB outperforms the MySQL database in both the write and read operations. Similarly, they have not described the database design, how they set up the indices on MySQL and what query they used for lookup of the data in their paper.

Many other similar studies [294] [295] [296] have also been conducted in order to compare the performance or basis attributes between RDBMS and NOSQL databases. However, the RDBMS and NOSQL databases are completely different types of database systems, so it is not equitable to compare these two different types of databases since RDBMS and NOSQL focus on different aspects; hardware, software, and data. As Nance et al. [205] point out, the RDBMS will not be replaced by NOSQL, because it is typically suited for those in line of business applications that are supporting business operations. On the other hand, the NOSQL database is better for serving large, public and content centric applications.

The main differences between our study and previous studies are:

1. As mentioned above, many studies of NOSQL have been conducted, but none in astronomy domains with the purpose of investigating the feasibility of using NOSQL databases to store and manage astronomical data. Therefore, this provided the opportunity to perform our study. In this chapter, we will not compare the RDBMS with NOSQL databases for astronomical data management; the main focus in this chapter is discussing the design for mapping the astronomical data into NOSQL databases, presenting some scenarios and examples for mining particular astronomical data from NOSQL, and evaluating the performance of inserting, querying and deleting the astronomical data from the NOSQL database.

2. Most of the existing NOSQL studies has not presented full design details of NOSQL databases; hence, we will use MOA as our case study and present full implementation details of NOSQL databases in this chapter.

In addition, all data used in this chapter are based on real astronomical observation data, which are provided by MOA.

6.2 Data Management Approaches: File System vs. Relational Database System vs. NOSQL Database System

The file system, relational database system (RDBMS) and NOSQL database system are three different types of data management systems. Each of these three data management systems use a different mechanism for data storage and data access, but the common purpose of these three systems is to offer the ability to manage and access data in an efficient manner. In this section, we will explore these three different data management systems, discussing the storage mechanisms and data access mechanisms used by these systems.

6.2.1 File System

The file system is the component of a computer operating system that provides hierarchical storage and organization of data. This was the main data management approach before the advent of database systems.

In this system, the data are stored as a “file”, with each file being a sequence of bytes that has other related metadata associated with it, such as file name, size and type, access control information and last modification date, among other metadata. After a file has been created, it will be placed in a directory (called a folder in a Windows OS). The directory is a special file that contains a list of other files (data files or sub-directories), and different types of data files are allowed to be stored in the same directory. From a data management point of view, all logically related data files should be stored in the same directory; each directory can be seen as a collection of logically related data files. For example, the observation images are stored in one directory, and the metadata of transients are stored in another directory.

All data files stored in a file system will be determined, accessed and interpreted by the programs, and each program has responses to define and manage its own data. This means that if we want to look up some particular data, we need to run a particular program and search through the particular directory starting from the first entry until we find what we want. In other words, data manipulation services provided by the file system are based on a collection of programs [159]. Figure 38 illustrates how to lookup data from a file system.

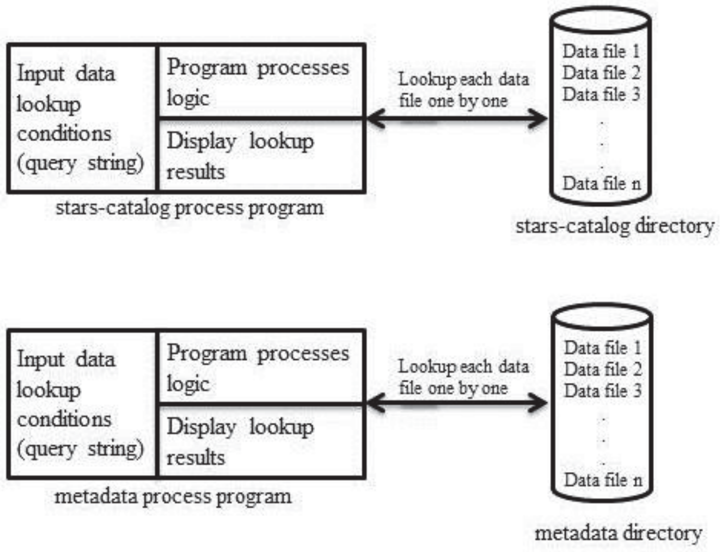


Figure 38: Diagram of lookup particular data from a file system. For example, if we want to search the nearest catalogued stars of a particular transient, a particular program (in this diagram is called stars-catalog process program) will be executed, searching data files in the stars-catalogue directory one by one until all matching results have been found, then forming the search results into a particular format, such as displaying the results in the program, storing them in another data file, or using them in a VOEvent packet. However, if we want to look up metadata of a particular transient, we need to run another program (metadata process program) and search through the metadata directory that contains all the metadata of all the transients.

There are two main advantages of the file system. First, the file system is very easy to use for managing data, because the file system is part of the operating system, so even end users do not have to have any programming skills; they will still be able to organize, access, and retrieve data through using UI (user interface) features provided by the operating system. Second, using the file system for data management does not require special technical persons to handle the database; programmers can easily write their own programs for data searches without learning any new technologies.

However, the disadvantages of the file system are greater than its advantages. Each disadvantage is described in the following points:

- **Data redundancy**: In the file system, the same data may be duplicated in more than one data file and we have no control over this data duplication (unless we check each data file line by line to remove duplicated data). In fact, the data duplication wastes a lot of storage space, because additional space is required for storage of the duplicated data. On large scale systems, data redundancy will lead to storage space filling up very quickly and will prove to be cost in-efficient. Moreover, and importantly, data redundancy can lead to loss of data integrity, which means that the data is no longer consistent.

- **Data dependence**: As mentioned above, in the file system, each program will manage and use its own data files and data stored in the file depends upon the program, which means that the structure of the data file is coupled with the program. This means that if we want to make any change to a data file, such as data type, size of data field and format, we have to modify the program as well.
  In addition, sometimes there might be more than one program accessing the same data file to perform different operations. In this situation, the programmer needs to identify all of the affected programs, modify them and re-test them. Obviously, this is a very time consuming task and modifications are likely to produce errors, especially in a large data system.

- **Data searching performance**: Data searching performance is another important issue for the file system, because the file system does not provide any indexing or optimization searching features for data management, so performance is really dependent on the search algorithm that is implemented in the program codes. When the structure of the data file has been changed, the programmer might need to optimize the search algorithm in order to maximize the search performance. In fact, optimizing the search algorithm is a very time consuming job, because sometimes more than one program might need to be optimized.

Moreover, as mentioned before, searching data in a file system starts from the first data file in the directory and runs until the desired data are found. If the result we want is in the last data file in the directory, then the program needs to go through all of the data files in the directory. This might not cause big performance issues when only a few data files exist in the directory, however, when there are a large number of data files in the directory the search performance will decrease significantly.

The file system is not a scalable data management approach due to lots of disadvantages and limitations as described above. Thus, this system is not suitable for use by those systems/projects that need to manage a large amount of data files, like LSST.

6.2.2 Relational Database System (RDBMS)

As there are many drawbacks to using a file system for data management, thus, the database model was developed. The database model is typically designed for data management purposes, and the first database was proposed by Edgar Codd [160] in 1970. He wrote a number of papers that outline the use of the relational database approach in data management. In a relational database, the data are divided into tables, with each table representing one subject of data. All data are accessed via tables [159], rather than via different formats of data file.

However, there is one important thing the needs to be designed before implementing an actual database and that is the database schema. This is the blueprint of how a database is constructed; it defines the objects (tables, views, stored procedures, etc.) that will be created in the database, how the data are organized, and how the relations among them are associated. Moreover, it also formulates all the constraints that are to be applied on the data. In other words, the database schema is the structure of the database.

Normally, the normalization process is a necessary step before designing a database schema. The normalization is typically a refinement process after the initial exercise of defining the necessary data objects that should be in the database. These data objects include; tables, columns within each table, relationships, and actual data that need to be stored. There are four major benefits that can be achieved through normalizing:

1. Ensuring data are stored in the correct table, so that the data are arranged into logical groups, with each group describing a part of the whole. This means that data are stored where they logically and uniquely belong;

2. Reducing data redundancy in the database;

3. Accessing and retrieving the data efficiently without losing data integrity; and

4. Reducing the probability of restructuring the database when new data is added.

Of course the database schema can be designed without normalization processes. However, without normalization, the database could be inaccurate, slow and inefficient. In the worst case scenario the database might not produce the data that we expect. For these reasons normalization processes are an important part of a successful relational database design.

Once the database schema has been designed, we will implement the database following the schema. As already mentioned, a relational database is a collection of tables. A table consists of two parts, the columns and the rows. A column (also referred to as the attribute) is name paired with the data type; each table may contain one or more columns. A row is also called a record or tuple, and the value of a row is the data instance. Similar to a column, each table may contain one or more rows; each row represents a set of related data, and every row in the same table shares the same structure.

After all of the required tables have been created, we need to establish the relationship between tables in order to pull the data together in a meaningful way. In other words, all tables that are logically related will be linked together [159]. The relationships between tables can be made through defining the primary key (PK) and the foreign key (FK) in the tables; each table should have at least one column defined as the primary key, and the system uses this primary key to access the table. When a primary key migrates to another table, it becomes a foreign key in the other table; the data type of the foreign key must be the same as the primary key in the related table.

There are three types of relationship design patterns supported in the relational database design; one to one, one to many, and many to many; with each pattern used to handle different relational situations. For example, in an astronomical database, we might have a `transientInfo` table and a `photometry` table. The `transientInfo` table would contain metadata information about transients, such as the transient ID, observation time and exposure time, and the primary key of this table will be the transient ID. The `photometry` table contains photometry measurement data of transients. As we know, a single transient will have lots of photometry measurement data, so the transient ID will be the foreign key in the `photometry` table and the relationship between the `transientInfo` table and the `photometry` table is one to many.

In fact, we might create a few individual and unrelated tables in the relational database to serve small, or independent, tasks in some situations.

Accessing and manipulating data held in the relational database must use a standard language called Structured Query Language (SQL); the SQL is a set of instructions that are typically designed for interacting with a relational database, and this language consists of three main components [159] [161]:

1. **Data Definition Language (DDL):** The DDL commands are used to modify the actual structure of a database rather than the contents of a database, such as creating a new table (CREATE TABLE), modifying the table structure (ALTER TABLE) and deleting an existed table (DROP TABLE).

2. **Data Manipulation Language (DML):** The DML contains the SQL commands that are most frequently used by the users, including retrieving information from a database (SELECT), adding new information to a database (INSERT), updating existing information (UPDATE), and removing particular information from a database (DELETE). In other words, the DML commands are used to manipulate the contents of a database.

3. **Data Control Language (DCL):** The DCL provides the ability to control user access to the database system in order to increase security and protect the database system. The DCL commands are the core of the relational database security model.

Moreover, we can improve data query performance by using the index feature provided by the database. Indices provide the ability to conduct rapid random lookups and efficient access to the recorded data, and they can be created using one or more columns of a table. With Indices, the required data can be quickly located without searching every row in a table every time a table is accessed.

Although the relational database system improves performance and greatly improves the efficiency of data management over the file based system, it still has disadvantages and lacks performance in some situations, as described below:

- **Complexity of database maintenance**: In a large data system, data will be divided into small subject tables, and there are many table relations that will be created in order to linked the tables together. Sometimes, a large relational database instance will contain more than a hundred tables and relations. However, a large number of tables and relationships will increase the complexity of database maintenance. In addition, we might add new columns onto existing tables in order to handle extra requirements or deal with future changes. In fact, adding new columns onto an existing table is also a complicated task, because we need to first remove all of the existing relations from the table that need to be modified, and then add the new columns to the modified table. Once all of the required columns have been added to the modified table, we need to re-establish the relationships with all related tables. In a large relational database, a single table might have more than one related table. Thereby, modifying the table structure is a very time consuming job and can easily produce errors.

- **Complex SQL commands**: When the relational model is complex, the SQL commands can be very complicated, which will increase the difficulty of writing SQL commands, or tracking SQL command errors. For example, a single data querying SQL command might include many joins (inner join or outer join) across many tables (at least three tables) and several nested conditions syntax, and the length of the SQL can also be very long (more than ten lines are possible) [162]. In this case, if the output of the query result is outside of our expectation, the developer will need to spend a lot of time locating and modifying the errors.

- **Fixed schema**: The database structure and data types are fixed in advance, so when a new data item needs to be added or we change the data type of a column; the entire database must be altered. In addition, the database must be taken offline during the database modification period.

- **Limited scalability for dealing with data increase demand**: The relational database is vertically scalable; in other words, if we want to scale the relational database for storing and processing large volumes of data, we need to boost the machine hardware where the RDBMS system is installed, which can become very expensive. Of course, it is possible to spread the RDBMS system over many servers, but this is very complex and very time consuming process, and additional engineering is required.

As the relational database has lots of limitations when dealing with a situation involving massive volumes of data, the NOSQL type database system has been proposed as an alternative data management approach. This type of database system is typically designed for scalable data storage and for resolving performance bottlenecks of relational databases. In the next section, we will explore and discuss this type of database system in detail.

6.2.3 NoSQL Database System

In the age of big data, the relational database model has faced scalability issues and performance issues as data volumes increase. Thus, NoSQL databases have been developed as an alternative data management solution to relational databases.

The term NoSQL initially referred to a database system that had no structure and no standard query language (SQL is standard query language for a relational database), but after a few years the developers found a more meaningful translation for this term; Not Only SQL. Any database system that does not use a relational model can be described as a NoSQL database.

The concept of NoSQL was proposed long ago, but the big movement in this area began in 2009 [163] and has since grown rapidly. This is because the NoSQL database system has been found to have advantages when dealing with large volumes of data. The main advantages of the NoSQL database system over the RDBMS system are:

- **High performance data access**: One design factor of the NoSQL database system is that it provides efficient data access in the large data volume context. In order to achieve this goal, the NoSQL database system uses BASE (Basically Available, Soft state, Eventual consistency) compliancy, instead of ACID (Atomicity, Consistency, Isolation and Durability) compliancy as in the RDBMS system [189] [190]. The main reason why the ACID compliancy of relational databases usually has worse performance is due to its rigid enforcement of data consistency. Thus, the NoSQL database system sacrifices ACID compliancy for performance. In other words, the NoSQL database system has no foreign keys/referential integrity constraints and no JOIN operations, as these features would significant affect performance.

The "consistency" that is a concern with NoSQL databases is found in the CAP (Consistency, Availability, Partition tolerance) theorem (also referred as Brewer's theorem [191]), which signifies the immediate or eventual consistency of data across all nodes that participate in a distributed database.

- **Flexible data model**: The NoSQL database system is based on a dynamic schema (also referred to "schema-less"). This dynamic schema nature allows the NoSQL database system to accept all structures of data (structured, semi-structured and unstructured) much more easily than in a relational database (which requires a predefined schema). Thus, the NoSQL database systems can support many use cases and others that do not fit well into a RDBMS system. In addition, we can modify the structure of a NoSQL database in real-time, which means that any system updates, modifications or adding of new information into the system can be made at any time without taking the database system offline.

- **Scalability**: The NoSQL database system is horizontally scalable, which means that data are very easily distributed across multiple servers. These multiple servers can be cheap commodity hardware or cloud instance, which is much more cost-effective than vertical scaling. Moreover, many NoSQL database systems also support distributing data across servers automatically, so sometimes NoSQL database systems are also referred to as cloud based databases [164].

- **High Data Availability**: All NOSQL databases achieve high data availability through using the replication feature; the data replication model of NOSQL databases consists of one master server (also called "primary") and one or more slave servers (also known as "secondary"). In this model, the can only be one master server at any particular time, with the other servers being slave servers. The data writing operation always takes place on the master server, but the read operations can be served by either the master server or the slave servers. When new data has been written to the master server, the same data will be cloned to all of the slave servers. In other words, all of the slave servers have a copy of the same data as the master server. Moreover, the replication feature also offers a fault-tolerance mechanism (also called fail-over) [266] to guarantee the high availability of the database systems. When the master server goes down, one of the slave servers will be automatically elected as the new master. Therefore, even if a single database server is unavailable, the whole database system is still able to continuously offer service to users without interruption, because users can access the data they want from other database servers (slaves) within the replication infrastructure. Please note that enabling the replication feature is an important condition to achieving high data availability; without data replication, the data availability of the NOSQL database is still not able to be guaranteed, because if only one NOSQL database server is utilized to provide services, and this database server is unavailable, users can only access the data until the database server comes back online.

Even though the NoSQL database system offers lots of remarkable benefits when handling ‘big data’ situations, it is not without drawbacks. In general, the NoSQL database system has two major drawbacks. First, the NoSQL database system does not have a standard query language as with the RDBMS system; each type of NoSQL database system has its own query language. In other words, developers need to learn many different query languages in order to use different types of NoSQL databases. This non standardization of query languages nature requires extra effort and time spent by the developers.

The second drawback is that the NoSQL database system does not have a standard interface; because each NoSQL database system has unique aspects in both how the data are stored, as well as how different bits of data relate to each other, no single interface manages them all. When using a new NoSQL database system for data storage, the developer must invest time and effort to learn about the new database system.

There are more than a hundred open source and commercial NoSQL databases available, but they can be divided into four major categories; Key/Value stores, Object Oriented stores, Document Oriented stores, and Graph stores [165]. Each category is described in detail below:

- **Key/Value stores**: The fundamental data model of Key/Value (KV) stores is based on an associative array (like a hashtable or map). To store data in a KV database, it is necessary to create a key for each record; a key consists of a string and cannot be duplicated. After the key has been created, the actual data that need to be stored is considered as the value; the value can consist of any primitive types (such as string, integer, or array), and it is retrieved through indexing a key. In other words, in a KV database, data is represented as a collection of key/value pairs, and each key is only allowed to appear once in the collection.

- **Object oriented stores**: Object oriented stores (also referred to as Object databases) first appeared in 1986. The data stored in an object database are formed as objects, as in object oriented programming.
  The main usage of an object database is in object oriented development areas. This category of database is mainly used for storing objects created by programming languages such as C++, Java and others. The object oriented stores databases works well for storing and managing multimedia data, such as images, text, audio and video.

- **Graph stores**: The graph database is a database management system with CRUD methods that expose a graph data model; it is typically designed for storing inter-connected data. In theory, the graph database consists of two components: “Node” and “Edge”. A node is used for data storage, with all data in the graph database stored in a series of nodes. An edge is a line that connects one node with another node. Each edge must connect with one node and it represents the relationship between the nodes. The graph database is good for shaping of the domain when it is naturally a graph (i.e. social networks) and is often used for storing and working with unstructured and semi-structured data (i.e. blogs, audio files, video files, emails, etc.).

- **Document Oriented Stores**: A document-oriented database is another type of NOSQL database that is designed for storing, retrieving and managing structured, semi-structured, or machine-readable data. The key concept of a document-oriented database is that it uses the document as its storage unit, instead of the records as in a relational database.
  In general, the documents inside a document oriented database can be any standard document formats, such as XML, YAML, JSON, BSON and flat text [166]. This category of database is useful for content management systems, blogging platforms, web analytics and real-time analytic systems.

Some of the major open source NOSQL database implementations are listed in Table 6:

Database name	**Data model category**	**Implementation language**	**License**
Oracle Berkeley DB (DBD)	Key/Value store	C	GNU Affero General Public License (AGPLv3)
Redis	Key/Value store	ANSI C	BSD
NeoDatis ODB	Object Oriented store	Java	GNU Lesser General Public License (LGPL)
Db4o	Object Oriented store	Java	GNU General Public License (GPL)
Neo4J	Graph store	Java	GPLv3
OpenCog	Graph store	C++	AGPL
MongoDB	Document Oriented Store	C++	AGPLv3
Xindice	Document Oriented Store	Java	Apache License 2.0
CouchDB	Document Oriented Store	Erlang	Apache License 2.0

Table 6: Major database implementations of four different NOSQL categories.

Each category of NOSQL database system has its own strong usage area; there is no one correct category that is suitable for all situations. As mentioned before, the NOSQL database is not designed to replace traditional relational database systems. Instead, it is designed for handling those situations that traditional relational database systems cannot cope with, such as out scaled data, requiring a dynamic schema to meet the demands of evolving data, and storage and management of large quantities of data [267][268][206][270]. The next section begins the discussion of how to use a NOSQL database to store and manage different astronomical transient data.

6.3 Suitable NoSQL Category for Transient Data Storage and Management

NOSQL databases are highly scalable data management solutions and have the ability to provide high performance data access in extremely high volume data situations. Thus, NOSQL databases could be a possible data management solution for astronomical data storage and management, especially those astronomical projects that will generate, and need to manage, large quantities of datasets.

In this chapter, MOA is used as a case study to discuss implementation details of utilising NOSQL databases for managing MOA's transient data, through presenting some data access use cases and discussing the performance of the NOSQL type data management approach.

6.3.1 Discussion of Different NoSQL Categories for Storing Transient Data

As mentioned in the previous section, each NoSQL category has its own strengths and weaknesses. Not all NoSQL categories are suitable for astronomical data storage, so an appropriate NoSQL category needs to be chosen before anything else is done. After careful consideration, it has been found that object stores, graph stores and key/value stores do not satisfy the requirements of the MOA project, either in terms of data storage requirements, or data access requirements.

First, the object store database is mainly designed to map to objects of abstract data types in existing code in some programming language that supports OO. However, our objective is to use NoSQL database to manage and store astronomical transient data, rather than some kind of objects. Thus, this type of database is definitely not a fit for transient data storage.

Second, all of our transient data are structured and machine readable data, but a graph store database is designed for dealing with unstructured and semi-structured data, rather than structured data. In addition, the relations within transient data are not well represented as a graph. Therefore, a graph stores database is also not suitable for transient data storage.

Third, a key/value store database has the ability to store structured and machine readable data. However, this category of database has a big drawback, which is that value contents (data) are opaque to the database itself and queries are key based. When a query operation is performed, the query input must be a key name, and the database will simply return the values (values could be string, integer, and etc…) that belong to the input key (working in the same way as retrieving data from a hash table, or a map in a programming language). In other words, there is no way to query based on the content of the value. However, in most situations, we only want to retrieve the data that matches the specific query conditions, rather than retrieving all data of a particular key. Thus, this category of database does not satisfy our data access requirement.

On the other hand, a document store database fully matches our data access and data storage requirements, due to the following three reasons:

- In document store database, data is stored in the form of documents and these documents are grouped together in "collections". Moreover, documents within the same collection can have completely different structures.
  The idea of document store is very similar to key/value store, because data stored in a document will associate with different keys, which is basically using the same

concept as key/value stores. However, the important difference is that a document supports complex data structures, since a document allows nested data (array) and sub documents to be associated with each key, rather than only having a primitive value as in a key/value store database. Rich data structure support makes a document store database more flexible and offers a strong ability to handle machine readable, structured and semi-structured data.

- The data model of document stores is very simple, so we do not need up-front logical modelling as with RDBMS; we can simply create subject based collections, transforming transient data into document format, and store them in the corresponding subject collection. The simple data model will also reduce the complexity of future database maintenance, as modifying the database structure (adding or removing documents or collections) can be done dynamically without touching any of the existing database elements (documents and collections). In addition, modifying the structure of a single document will not affect other documents in the same collection. The most important thing is that everything can be done on the fly, and this is a good way to deal with schema evolution.

- The internal storage mechanism of document store database embeds attribute metadata associated with the stored content, which essentially provides a way to query the data based on the contents. This is a very important feature for an astronomical database.

Thus, we finally decide to utilize a document store NOSQL database in this research project for transient data management.

6.3.2 Choice of Document Stores Implementation

The XML, JSON and BSON formats are common standard data storage formats used by document stores database products. In the early stage of the development, a document stores database called Xindice [167] was used for our data storage. Xindice is an open source database developed by Apache, and data is formed as an XML document to store in this database, so it is also referred to as an XML based database.

Xindice allows a different structure of XML document to be stored in a same collection, because it does not require an XML schema association with the collection; it is referred to as being schema independent [168]. However, after mapping the data into XML documents and storing these XML documents into Xindice, several disadvantages were discovered in using XML documents as data storage units, as well as other disadvantages of Xindice.

First, the XML syntax is redundant; this redundancy may affect application efficiency through higher processing and transmission costs.

Second, XML documents are very verbose in comparison to other alternative ‘text-based’ data formats, such as JSON and BSON, because XML requires all open and close markup tags to be present in order for it to work properly; these open and close markup tags will also consume lots of extra storage space.

Finally, XML documents are stored in Xindice via document trees (this storage mechanism is common for all type of XML based databases); when running a lookup on some particular data, the search must go through all branches of the tree before it locates the expected results. If the structure of the XML document is complex and there are a large number of documents in the same collection, the querying performance can be very slow. In fact, in our situation, each collection could contain a large number of XML documents, and the structure of some of the XML documents could be very complex (many nested tags) after data mapping. Thus, XML based databases do not fit the requirements of our project, and it is not a scalable data management solution for any future astronomical projects, so XML based databases are rejected and it is necessary to look for other types of document stores databases.

Following further research, another type of document stores database is found, which is called MongoDB [169]. MongoDB has been developed and maintained by 10gen (10gen has now changed its name to MongoDB Inc.) since 2009; this database is an open source database written in C++ and licensed under GNU Affero General Public License [170]. It is the fourth most popular type of database management system [171], and the most popular document stores database with a large number of user groups, strong community support and wide use by a lot of different projects, including The New York Times, Foursquare, SourceForge and eBay, among others.

Using MongoDB for data storage has many advantages over XML based databases. The major benefits are:

- **Lightweight and efficient data format**: The data storage format used by MongoDB is BSON (Binary JSON) [172]; it is a binary representation of JSON documents, with the data structure of BSON documents composed of field-and-value pairs and using "," to separate different pairs, as seen in the example below:

  {"field1":"value", "field2":"value"}

  The BSON format is much more lightweight compared to the XML format, because it does not require open and close markup tags to make documents work properly, so the data structure of the BSON format is clear and robust. The most important thing is that, without the need for overhead markup tags, this BSON will save lots of disk storage space. In fact, reducing overheads to a minimum is an important factor for any data representation format. Moreover, BSON is a very efficient data format; because it uses C data types, encoding data to BSON or decoding data from BSON can be performed very quickly in most programming languages.

- **Rich queries**: The MongoDB query language aims to offer similar features to SQL; it supports both complex queries and ad-hoc queries [270]. The MongoDB query engine allows the user to search for data from documents through using field names, data range, regular expression, geospatial and even full text searches, which makes it much more powerful than XPath. With this powerful query engine, we are able to search data from MongoDB with multi query criteria. In addition, The MongoDB query engine has a built-in 2D search feature [271], which is a very useful feature in astronomical contexts, because in some situations, we may want to list event data by the given specified sky coordination, and 2D search feature is well suited for this particular data search situation mentioned above.

- **Indices**: MongoDB supports creating unique indices for enhanced query performance over a RDBMS system. Indices can be created on single and multiple fields, these indices are B-tree indexes and defined per collection, which means each collection has its own indices. Moreover, we can also enhance query performance by putting only documents of a single type into the same collection.

- **Replication and auto failure recovery**: MongoDB also supports replication (also referred to as ReplicaSet), which allows us to use a set of servers that are linked in a master-slaves mode. Setting replication in MongoDB is easy and fast. Furthermore, the failover mechanism of replication is automatically handled. If the primary server has a problem and goes down, the secondary server will become the primary automatically without the need for any human interaction. Note that a minimum of three servers are required for replication; Primary server, Secondary server, and Arbiter server. The Arbiter server is not used for data storage, but is only used during failover to decide which server will be the next primary server. Indeed, the election process for the new master server will be performed automatically and instantly when the old primary server is unavailable.

- **Auto sharding**: Sharding distributes the data across physical partitions to overcome issues of hardware limitations (CPU, storage space and RAMs), and the data is automatically balanced in the clusters.

- **Rich programming language APIs:** MongoDB supports a variety of programming APIs that allow programmers to develop data access programs with their familiar programming languages. These APIs cover almost all modern programming languages, including C++, Python, Java, Javascript, PHP, C, and so forth.

- **Binary communication protocol**: The communication protocol used by the MongoDB is called Mongo Wire Protocol [173], which is a simple socket-based, request-response style protocol. With Mongo Wire Protocol, programs can communicate with MongoDB through a regular TCP/IP socket.

- **Cost effective**: MongoDB is also a cost effective data management solution [275], because it uses a horizontal scaling approach to serve the increased workload and increased dataset quantities. Additionally, horizontal scaling could reduce the cost in terms of hardware and storage, because increasing the capacity of the database system can be done through using cheaper commodity machines, rather than purchasing expensive hardware (RAM, HDD, etc…) to boost the capacity of the single server machine.

Even though MongoDB offers lots of advantages in data storage, data access and data management, similarly to other database systems, MongoDB also has its drawbacks.

First, a BSON document has size limitations; a single BSON document cannot exceed 16MB, and any BSON document over 16MB will not be able to be stored in MongoDB.

Second, NOSQL type databases do not implements or supports the 'join' feature, so MongoDB only allows a query on one collection at a time. As mentioned before, different types of data will be stored in different collections. Therefore, if we want to display multiple types of data in a single output, it is necessary to make multiple queries to gather different types of data from different collections, and then combine them manually within the code. In other words, the extra programming efforts need to be done in the program side.

In fact, the lack of join is the major drawback in migrating from SQL to NOSQL. However, none of the database management solutions are perfect; extra coding efforts are costs that the programmer needs to pay when they try to use benefits offered by NOSQL databases.

However, the disadvantages of MongoDB are relatively small compared to advantages provided by MongoDB, and the MongoDB is a feasible NoSQL solution for astronomical data management, because it shows a strong ability to store and manage transient data in an efficient manner. Thus, we finally decided to use MongoDB database as the data management system. It should be noted that there is another similar database implementation called CouchDB, which uses JSON as its data storage format. However, the data access mechanism supported by CouchDB is limited, since it only allows data exchange through REST API. Therefore, the CouchDB has not been considered to use for our data management system.

6.4 Design & Implementation of NoSQL Based Astronomical Database

MOA has lots of different types of data, each type of data contains many data files and these data files are currently formed in flat file format for storage in a file based system. Some examples of the types of data used by MOA are summarized in Table 7 and will be discussed in detail in this section.

Data type	Comments
Exposure metadata	Fits image headers
Reference imaging list	Recording the best image in each field-colour-chip combination, used for difference imaging analysis
Astrometric calibration	Astrometric calibration coefficient derived from reference list, used for calculating RA and Dec of new transient
R stars catalog	Used for extracting the nearest catalogued stars of the transient
Event portfolios	Storing data of interesting transient events

Table 7: Existing datasets that need to be stored in MongoDB.

As discussed in the previous section, MongoDB uses BSON documents as the data storage format, so we need to reformat all of the transient data from a flat file format to BSON document format in order to store transient data in MongoDB. This section will discuss our database design in detail, presenting a Java code example for creating BSON documents, and BSON document examples for each type of data listed in Table 7.

6.4.1 Database Structure Design

The database structure design in MongoDB are different from relational database design. There is no need to create tables, defining relationships between the tables, or any normalization processes, because MongoDB is not a relational based database. What is required is to group the same type of data in the same collection in order to make the database structure more logical. In other words, each collection will only represent one type of data.

There are two ways to creates a collection in MongoDB; either using a MongoDB shell command `db.createCollection(collectionname)`, or using a MongoDB API driver to create collections programmatically. However, using a shell command to create collections is very cumbersome, because the *`db.createCollection`* command will only create one collection at a time. For example, if we need to create seven collections in our database, the same shell command needs to be repeated and entered seven times. Therefore, we will use the programmatic method to create all of the collections in MongoDB.

The MongoDB API driver needs to be utilized in order to create collections in MongoDB programmatically; without the MongoDB API driver, it is not possible to communicate with MongoDB in the program. In our example, the Java programming language is used to create a list of collections in MongoDB. The Java version of MongoDB driver API can be downloaded from the MongoDB website [174].

After the MongoDB Java API has been downloaded and included as a program dependency, the program needs to connect with MongoDB instance before creating collections in MongoDB. Establishing the connection operation can be done through using a `MongoClient` object and `MongoDatabase` object, as shown in the code example 18:

```java
MongoClient client = new MongoClient("locahost",27017);
MongoDatabase database = mongoClient.getDatabase("moa");
```

Code example 18: Establishing the connection to MongoDB.

At this point, the `MongoDatabase` object will be a connection to a MongoDB instance for the specified database (each MongoDB instance can contain many databases). Next, the `createCollection (String collectionName)` method provided by the `MongoDatabase` object can be used to create a collection in MongoDB. The code example 19 below show how we create a list of collections in MongoDB:

```java
String[] collectionName =
         {"exposuremetadata","referenceimages","astrometriccalibration",
          "photometriccalibration","eventportfolios"};

for (int i = 0; i < collectionName.length; i++)
{
    database.createCollection(collectionName[i]);
}
```

Code example 19: Create a list of collections under “`moa`” database in MongoDB.

After the above codes are executed, all of the necessary collections for the case study have been created. Figure 39 illustrate the database structure of MongoDB database:

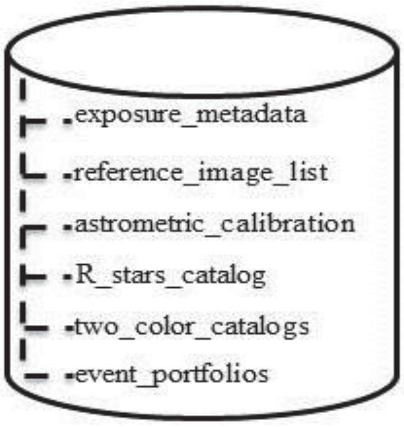


Figure 39: The database structure of MongoDB database. As mentioned before, the MongoDB is schemaless database, so the database structure of MongoDB is very simple and straightforward, when more types of data need to be stored in the database, we only need to creates a new collection within database without modifies existed database structure.

6.4.2 BSON Document Design

The important part of document design is reformatting data from a flat file format to a BSON document format. In fact, each type of data contains lots of data files (in flat file format), so if we simply put all of the data files of a single type into one BSON document, the size of the BSON document will definitely be over 16 MB. Moreover, the BSON document structure will be very messy, not robust, and hard to read. Thus, we decide to form each data file as a single BSON document. The benefits of this approach is that the BSON document structure will be very clear, robust, easy to read by machine, and will allow for easy updating of existing data or insertion of new fields into a BSON document when necessary.

However, as discussed above, each BSON document represents a single data file, so there are a large amount of BSON documents to be created and stored in MongoDB. In fact, using a shell command to perform this task is very inefficient since a MongoDB shell command only allows the creation and storage of one BSON document at a time, which is the same as creating a collection in MongoDB. Thus, the programmatic approach is still the best solution.

The remaining part of this section will present a BSON document example of each type of data file shown in Table 6 and example Java codes for reformatting these data files from the original flat file format to the BSON document format.

Exposure metadata

The exposure metadata are FITS image headers. All of the FITS images are captured by the MOA camera on a 1.8 metre telescope. As discussed in Section 4.1.1, there are 10 raw images (FITS format) captured by a single exposure, but metadata of these 10 raw images are the same. Therefore, each exposure set only has one metadata stored in the MongoDB in order to avoid duplication.

All of the exposure metadata are currently stored in flat files called “info”, and each “info” file contains all of the header metadata of a single FITS image. FITS image metadata consists of a list of key/value pairs, with each value associated with one data keyword, which is very similar to the BSON document format. So, after reformatting the exposure metadata as a BSON document, the data keyword will be formed as the field name in the BSON document, and the value of each data keyword will be formed as the field value in the BSON document. Reformatting exposure metadata to BSON document format through using MongoDB API driver is very easy, and only requires four steps.

First, a connection is established with MongoDB instance, as for creating a collection in MongoDB; then a `MongoCollection<Document>` object is created, and the `getCollection` method provided by `MongoDatabase` is used to specify a collection to operate upon. After the `getCollection` method has been invoked, the `MongoCollection<Document>` object will be connected to a specified collection.

Second, the `Document` object is created. The `Document` object is a representation of a document as a `Map`, with each `Document` object representing a single BSON document and being the major object used to create that BSON document.

Third, it is necessary to read all of the data keywords and keyword values line by line from an “info” file through using the I/O feature, and then store each data keyword and keyword value in the `Document` object using the `append(String key, Object value)` method provided by the `Document` object. The `append` method accepts two parameters; the first parameter is used to specify the field name in the BSON document, and the second parameter is used to specify the field value in the BSON document.

Finally, invoking the `insertOne` method provided by the `MongoCollection` object to store the completed BSON document in MongoDB. The JAVA codes for creating the BSON document are as shown in the code example 20:

```
String str = "";
Document doc = new Document();

while((str = br.readLine()) != null)
{
     // skip the comment line
     if (str.indexOf("#") < 0 && str.indexOf("COMMENT") < 0)
   {
        //get data keyword of exposure metadata
        String dataKeyword = str.substring(0, str.indexOf("="));
        dataKeyword = dataKeyword.trim();
        //get data value associates with data keyword
        String keywordValue = str.substring(str.indexOf("=") + 1,
                             str.indexOf("/"));
        /*Store data keyword and value in document object,
        first parameter will be field name of BSON document
        and second parameter will be field value*/
        doc.append(dataKeyword, keywordValue);
   }

   /*after all data keywords and data values has been
     stored in Document object, storing to MongoDB*/
   collection.insertOne(doc);
   //close connection
   mongoClient.close();
}
```
Code example 20: Java code for generating a BSON document.

The BSON document structure after the above Java code executed is shown below:

```
{
  SIMPLE: "T",
  BITPIX: 16,
  NAXIS: 2,
  NAXIS1: 2560,
  NAXIS2: 2048,
  EXTEND: "T",
  BZERO: 32768,
  OBSTEL: "MOA-II telescope",
  CAMERA: "MOA-cam3",
  FNUM: "f2.91",
  OBSVAT: "Mount John",
  LOGITUD: -170.467,
  LATITUD: -43.983,
  HEIGHT: 1029,
  DATE: "05-10-15T08:22:58",
  CLOCK: "GPS",
  TIMESYS: "UTC",
  TIME-OBS: "08:20:12.0",
  DATE-OBS: "2005-10-15",
  JDSTART: 2453658.847361,
  JDEND: 2453658.848521,
  EXPTIME: 100,
  RA: "17:47:30.0",
  DEC: "-34:15:00.0",
  EPOCH: 2000,
  AZI: -85.2,
  ALT: 48.9,
  ROT: 126.5,
  AIRMASS: 1.327698,
  DOMEAZI: 2765,
  FOCUS: -1.15,
  RUN: "B300",
  FIELD: "gb1",
  COLOUR: "R",
  SET: "a",
  CHIP: 0,
}
```

Code example 21: BSON document structure of exposure metadata.

However, the above code will only store one BSON document in MongoDB at a time. In fact, in some situations, we might want to store multiple documents into the database at a time. Fortunately, MongoDB API driver provides a method called `insertMany` that allows us to perform this task in a convenient manner, and the `insertMany` method accepts a `List` type object as the method parameter.

The only change that needs to be made on the above code is to store each `Document` object in some kind of `List` object (i.e.: `ArrayList`), rather than invoking the `insertOne` method to store the `Document` object in MongoDB immediately. Once all of the required `Document` objects have been generated and stored in the `List` object, it passes the `List` object to the `insertMany` method to insert multiple documents at a time into MongoDB.

It should be noted that the codes for creating a BSON document programmatically are common for all types of data, so the codes for creating the BSON document will not be presented in this sub section again, but we will present an example BSON document of a different type of data.

Reference image list

The reference image list file records the file name of the best seeing images for all field-colour-chip combinations and there are a total of 220 lines of data stored in this file (22 observation fields, each field is divided in 10 CCD chips and each line represents the file name of the best seeing image of one field-colour-chip combination).

This reference image list file is mainly used for difference imaging analysis. When new imaging has been captured after an exposure, the difference imaging analysis software will read this file to lookup the file name of the corresponding best seeing image. It then subtracts the best seeing image from the new image in order to identify new transient events.

The important elements that will be used by the difference imaging analysis software is the run number of the best seeing image, observation field, filter colour (mostly “R”) and chip number, so the BSON document structure consists of these four elements, and the field value of this BSON document can be obtained by splitting the file name of the best seeing image, because the file name of the best seeing image is constructed in a run number-field-colour-chip combination format.

There are two strategies that can be applied for designing BSON document structure of reference imaging list. The first one is grouping all field-colour-chip combinations into one BSON document through using a sub document feature. However, with this design strategy, the entire BSON document structure will be very complicated, messy and long. Additionally, the command to modify the structure of a single sub document is a little bit complex, so making any future changes to the sub document structure may require extra effort. Thus, this document design strategy is not suitable in this case. The BSON document example of the first strategy is shown below:

```
{
  Reference_list:[
                  {
                   field: ”gb1”,
                   colour: ”R”,
                   chip: 1,
                   run: “B1455”
                  },
                  {
                   field: ”gb2”,
                   colour: ”R”,
                   chip:  3,
                   run: “B1460”
                  },
                  {
                   field: ”gb10”,
                   colour: ”R”,
                   chip:  10,
                   run: “B1525”
                  }
                 ]
}
```

Code example 22: BSON document structure of the first design strategy.

The second strategy is that each field-colour-chip combination is formed as an individual BSON document; this document structure design strategy offers much more flexibility than the first strategy. With this strategy, the structure of each BSON document will be very simple and easy to read, and making changes to the document structure can be done with a simple command. When a single document structure needs to be changed, it is possible to directly modify the target document without touching any other documents in the same collection. The BSON document example of the second design strategy is illustrated below:

```
{
  field: "gb1",
  colour: "R",
  chip: 1,
  run: "B1455"
}
```

Code example 23: BSON document structure of the second design strategy.

Moreover, one important rule of design data representation format is to make it "as simple as possible", and the second design strategy fully follows this rule.

Astrometric calibration coefficients

In the current system, all 22 fields of astrometric calibration coefficients are stored in ".par" files under the ".par" directory, each "par" file is an individual coefficient table and containing 10 parameters of data, which are "ra", "dec", "sx", "sy", "x", "y", "mu", "nu", "phi", and "psi". These data are used for calculating RA and Dec of a new transient event, and this is a non-standard WCS (World Coordinate System) astrometic calibration that MOA has employed.

Each observation field is divided into ten CCD chips, and each CCD chip is also divided into eight sub frames. Each subframe has its own set of calibration constants described above. When a new interesting event has been identified, an analysis program uses the field-chip-subframe combination to select the corresponding astrometric calibration coefficient table, and uses parameters data stored in that table to calculate the RA and Dec through a mathematical formula (this calculation process is not the subject of this research).

Each "par" file should be formed as an individual BSON document, because each "par" file is an individual astrometric coefficient table. The BSON document structure consists of astrometric coefficient parameters data, field name, chip number and the actual sub frame from which the coefficients were derived. An example of an astrometric calibration coefficient document is illustrated in code example 24:

```
{
  field: “gb1”,
  chip: 1,
  subframe: 0,
  ra: “17:44:41.853”,
  dec: “-33:37:27.47”,
  sx: 1,
  sy: 1,
  x: 512.607045092382,
  y: 485.063814467840,
  mu: 0.578527269265222,
  nu: 0.578327173202944,
  phi: 0.150172954199544,
  psi: -1.136699314589505
}
```

Code example 24: BSON document structure of astrometric coefficient.

R stars catalog

The R stars catalog is a list of photometric measurements of all resolved stars on the reference images. These images were observed in a red passband filter designated “R”. The stars catalogue of each field-colour-chip combination are stored in a “.dat” file, with each line in the file describing all of the data (i.e. star id, xcenter, ycenter, ra, dec, etc.) of a single star, and there are large numbers of stars that are associated with a single field-colour-chip combination. Code example 25 shows the BSON document structure for a star in the gb5-R-3 combination file.

```
{
  field: “gb5”,
  color: “R”,
  chip: 3,
  starid: 47,
  xcenter: 634.596,
  ycenter: 5.015,
  mag: 13.348,
  merr: 0.040,
  msky: 1194.6770000,
  niter: 4,
  sharpness: 0.062,
  chi: 3.135,
  pier: 0,
  perrors: “No_error”,
  magfit: 0.0532055077,
  magres: -0.0132055077,
  ind: 1,
  ra: “17:50:56.712”,
  dec: “-29:18:12.61”
}
```

Code example 25: BSON document structure of R star catalogue, similar to the reference image list, each line in the “.dat” file will be formed as an individual BSON document;

A common operation MOA performs is to read this catalog and extract a list of nearest stars to a newly detected transient event. If we store star documents of all combinations into one collection as previously, the collection will contain large volumes of documents, and performing extractions of the nearest stars' operations could be inefficient. This is because, when extracting the nearest catalogued stars of an interesting object, we know which field that we need to look up, so we should not search star catalogs in other fields.

Therefore, the database structure (Figure 40) needs some modification, which involves removing the `R_star_catalog` collection by using the `db.Rstarcatalog.drop` command, and then creating collections for each field. After all of the field collections have been created, the BSON documents are stored into their corresponding field collection. For example, the star document of the gb5-R-3 combination above will be stored in the "gb5" collection. The new database structure is illustrated in Figure 40:

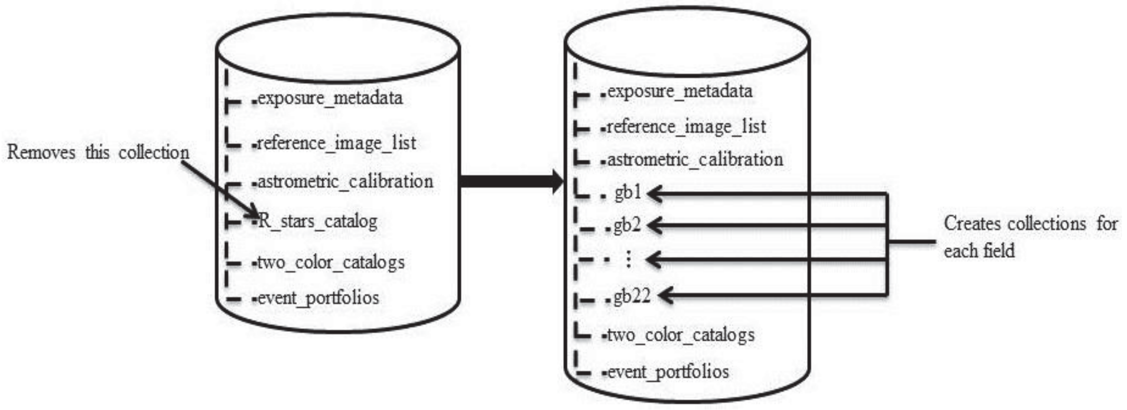


Figure 40: The left hand image is the database structure before modification, and the right hand image is the new database structure after modification.

Of course, we can apply the same design pattern in a relational database for holding R star catalogued data, which creates 22 tables, for which each table only stores one field of catalogued stars' data. However, this design pattern will lead to a more complex relational database structure, because it will require a lot of extra tables and relationships. If the data structure of an R star catalogue file changes (i.e.: adding a new parameter), we will need to modify the table structure of all 22 tables. In other words, using this design pattern for holding R star catalogue data in a relational database will increase complexity of database maintenance significantly. In addition, from a normalization point of view, a well-designed database should not have repeating sets of columns (in our case, all 22 tables will have same set of columns), same kinds of values should be placed in one table.

Thus, it is necessary to store the entire stars catalog data in one table (each star data will be stored in one row in the table), so the table will contain a large number of rows. As a result, the SQL syntax for extracting the nearest catalogued stars could be very complicated (the syntax comparison between SQL and MongoDB will be discussed in the next section), and query performance could also be very inefficient.

As can be seen, in the MongoDB context, we can easily create extra collections to store different field's stars data, modifying the existing database structure is simple and quick, and everything can be done in real time without affecting other collections in the database. These are one of the important reasons for deciding to use a NOSQL database as our data management system instead of a traditional relational database.

Event portfolio

The event portfolio is used for storing data of interesting transient objects and alerted objects. It is essentially a machine readable description of the event together with all relevant data. We follow the event portfolio concept similar to that of SkyAlert [7].

Each event portfolio represents a single interesting transient object and contains all the related data about this interesting object, the data stored in an event portfolio document including the object domain, run number, field name, colour, chip, RA, Dec, nearest catalogued stars, and the link to any external data (photometry, FITS images, and so forth). Unlike other types of data, the event portfolio document will only be created when new interesting objects have been found; in other words, the event portfolio document is created on the fly.

In some situations, the Paczynski model parameters (as discussed in Section 4.2.1) of some interesting transient objects might be updated or changed when more data is obtained. It would be optimal to record all these changes in the event portfolio document, rather than replace the old model parameters with the new ones, because tracking changes of the model parameters are very useful for future analysis.

Fortunately, we can easily achieve this goal using the sub document feature provided by the BSON document, because the BSON document allows for having the document as the field value and there are unlimited amounts of sub documents that can be associated with a single field. In other words, when Paczynski model parameters update, or the model changes, the new parameters of the Paczynski model will be appended to the `models` field in the corresponding event portfolio document, as shown in code example 26.

```
//open database connection
MongoCollection<Document> collection =
database.getCollection("eventportfolios");
Document subdocument = new Document();

//creates sub document for stores new parameters of paczynski model
Subdocument.append("tmax","tmax value");
Subdocument.append("te","te value");
Subdocument.append("u0","u0 value");
Subdocument.append("Ibase","Ibase value");

/*construct update syntax for appends sub document to models
field,the second parameter of updateSyntax Document object specifies
name of event portfolio that needs to be updated*/
Document updateSyntax = new Document("$push", new Document("2015-BLG-
                                  003$models", subdocument);
//performs update operation
Collection.update(new Document(), updateSyntax);

//close database collection
```

Code example 26: Example of appending a sub document to the Paczynski model field as a field value.

The event portfolio document for transient object “MOA 2015-BLG-003” is shown in code example 27.

```
{
  internal_id: “gb5-R-9-87873”,
  moa_id: “2015-BLG-003”,
  discovery_date: “2015-02-10T23:24:54”,
  ra: “17:54:17.55”,
  dec: “-29:00:25.38”,
  domain: “BLG”
  run: “B48266”,
  sequence: 87873,
  ccdx: 1142.7,
  ccdy: 3781.62,
  nearest_catalogued_stars:[
                             {
                               star_id: 161347,
                               ra:  “17:54:17.68”,
                               dec: “-29:00:26.32”,
                               mag: “17.53±0.14”,
                               separation: 2.39
                             },
                             {
                               Star_id: 161418,
                               ra: ”17:54:17.55”,
                               dec: “-29:00:30.09”,
                               mag: “16.13±0.12”,
                               separation: 4.30
                             }
                            ],
  models:[
           {
             tmax: ”JD2457081.66±0.22UT(2015-Feb-28.16)”,
             te: ”4.93±0.09 days”,
             u0: “10.3933±0.0789”,
             Ibase: “7.45±0.35”,
             modeltype: Paczynski
           },
           {
             tmax: ”JD2457063.84±0.00UT(2015-Feb-10.34)”,
             te: ”10.04±0.74 days”,
             u0: “0.0000±0.0006”,
             Ibase: “19.22±0.21”
             modeltype: Binary
           }
         ],
   external_photometry: “$DATAHOME/ephot/phot-gb5-R-9-87873.dat”,
   finder_chart: “$DATAHOME/datab/finder-gb5-R-9-87873.fit.gz”
}
```

Code example 27: Event portfolio for alerted object “MOA 2015-BLG-003”. Note that this document is only used for the purpose of demonstrating the event portfolio format, so this example document does not include all the related data; the parameter values of the second sub document in the models field are assumed values.

The use of sub documents in the `models` field allows us to update the model parameters as new data are obtained. In our system, when the new model parameters are updated, new model parameters are appended to the `models` field. In addition, this `model` field also allows us to add other types of models, such as binary lensing, though the Paczynski model is the most common.

In fact, it might be necessary to include more related data in the event portfolio document in the future; performing this task in a relational database is very complex and time consuming. First, we need to remove all relationships from the event portfolio table. Second, we adding the new column into the event portfolio table. Finally, we re-establish the relationships with all other related tables.

However, in the MongoDB context, it is only necessary to modify the codes of the event portfolio document creation program to include the new field and field value, and then run the same program to store the new format BSON document into the event portfolio collection. The structure of the other existing documents in the event portfolio collection will not be affected, or require any modification. Using a document based database provides the best flexibility to deal with this type of schema evolution.

Other type of data

There are lots of other types of data that can also be stored in MongoDB; for example, cross referenced OGLE stars and photometric calibration coefficients, among others. The same method as discussed above can be utilised to design the document structure for these data, so there will not be much discussion of the design details here in order to prevent repetition.

As mentioned above, the design method that we used for the document structure design can be applied to all types of data, so even if MOA has new types of data that need to be stored in MongoDB in the future, the developers just needs to follow the design method discussed in this section to create a new collection and document for dealing with the new type of event data.

The next section will discuss some data exploring scenarios, providing some data query examples, and comparing query syntax between MongoDB and SQL.

6.5 Database Querying Scenarios

Exploring data is a very important part of database manipulation. Every database product provides a data querying engine for retrieving specified data from the database. This chapter introduces some data exploring scenarios used by MOA and provides query examples of these scenarios.

6.5.1 Data Exploring Scenarios

The MOA has lots of data lookup scenarios that can be outlined, however, not all of the data lookup scenarios will be listed in this section, but some scenarios that are related to generating VOEvent packets will be. The main purpose of this section is to provide guidelines of how to use the MongoDB query engine for querying specified data from the MongoDB. A simple data query example will be provided as the starting point, and there will follow a discussion of some advanced data query examples in the latter part in this section.

We will only discuss data lookup scenarios and MongoDB query syntax to accomplish these scenarios in this section. The performance evaluation of MongoDB database will be discussed in Section 6.6.

Scenario 1: List all exposure run numbers for specified field-colour-chip combination

In some situations, the database user might want to explore all run numbers of exposure that are captured by a particular field-colour-chip combination (i.e.: gb5-R-0). To find this information, the query requires “field name”, “colour” and “chip” as the query parameters, and we can simply use the MongoDB command shell to complete this scenario. The MongoDB query syntax for this scenario is shown by the code example 28:

```
db.exposuremetadata.find({ field : "gb5", colour : "R", chip : 0 });
```

Code example 28: MongoDB query syntax for listing all run numbers of a specified field-colour-chip combination.

As we can see from code example 28, the MongoDB query syntax is based on BSON format, but it is very straightforward and easy to write, and it uses comma to separates multiple search conditions. However, most of the time, the query results will be used for some further analysis and calculations, rather than just for browsing the query results. Thus, it is better to perform a query operation programmatically instead of using the MongoDB command shell.

Similar to creating collections or inserting documents in MongoDB, the MongoDB API driver is utilised, but the codes are very simple and it is not necessary to enter the entire query syntax, as needed for querying data through using the MongoDB command shell. As previously mentioned, a MongoDB API exists for a variety of programming languages. Code example 29 below shows a Java example of programmatically querying data from MongoDB.

```
public void findQueryResult()
{
  MongoCollection<Document> collection =
                                  getCollection("exposuremetadata");

  Document query = new Document ("field", "gb5")
                              .append("colour", "R").append("chip", 0);

  //Search data from mongodb and stores result in iterable object
  FindIterable<Document> iterable = collection.find(doc);
```

Code example 29: Querying run numbers of specified field-colour-chip combination with a Java program

After the above codes are executed, the MongoDB API driver will construct and send the same query syntax as shown in code example 29 to the MongoDB server; the query results are then stored to the `FindIterable<Document>` object. The `FindIterable` is an iterable object that yields documents, which is a similar concept to the `ResultSet` object in the SQL context. Code example 30 shows how to use the `FindIterable` object for retrieving query results.

```
iterable.forEach(new Block<Document>()
{
   @Override
   public void apply(Document document)
   {
     //Display results, print out value of run number,
     //field, colour, and chip
     System.out.println("run number: " + document.getString("run"));
                        + " field: " + document.getString("field")
                        + " chip: " +document.getInteger("chip"));
   }
});
```

Code example 30: Reading query result, using the `foreach` method to iterate the results one by one, and applying a block to each resulting document.

Scenario 2: Lists metadata for specified reference image

It might be necessary to look at the metadata for reference imaging when performing difference imaging analysis operations. As discussed in Section 6.3.2, the reference image list and exposure metadata are stored in two separate collections (exposure metadata collection and reference image list collection), therefore, this query scenario will operate upon these two collections.

However, unlike SQL, it is not possible to accomplish this scenario in one query syntax, because a NoSQL type database does not support cross collection searches. Thus, this query scenario must be performed through two individual queries; the first query will search the run number of the reference image by a given field, colour and chip as the input parameters, and the second query will utilize the run number retrieved in the first query to search the metadata of the reference image from the exposure metadata collection. The MongoDB query syntaxes for this scenario are illustrated in code example 31:

```
db.referenceimagelist.find({ field : "gb5", colour : "R", chip : 3 })
db.exposuremetadata.find({ run : "B1447"})
```

Code example 31: MongoDB query syntax for lookup metadata of a specified best image. The first line in MongoDB query syntax is used for get run number of best image from the `referenceimagelist` collection. Afterwards, we can use a second line query syntax to lookup best image metadata for the gb5-R-3 combination from `exposuremetadata` collection.

As can be seen from code example 31, the MongoDB requires two individual queries to complete this scenario, but the query syntax are still relatively simple, so even if a semantic error exists in the query syntax, programmers are able to track and fix these errors quickly.

```
MongoCollection<Document> collection =
getCollection("referenceimagelist");
Document query = new Document("field","gb5")
                     .append("colour", "R")
                     .append("chip", 3);

//Performs first query
FindIterable<Document> iterable = collection.find(doc);
iterable.forEach(new Block<Document>()
{
   @Override
   public void apply(Document results)
   {
    //searching metadata of best image from exposuremetadata
    //collection via using run number get by first query
    MongoCollection<Document> collection =
    getCollection("exposuremetadata");
    //get run number from first query result
    //and use it in second query
   Document lookupMetadata = new Document("run", results.getString("run"))

    FindIterable<Document> iterable = collection.find(lookupMetadata);
    iterable.forEach(new Block<Document>()
    {
       @Override
       public void apply(Document metadata)
       {
         //Display observation date, ra, dec
         //for given run number. We can also use
         //metadata.toJson() method to read all metadata
         System.out.println("observation date at: " +
                          metadata.getString("date") +
                          ", ra: " + metadata.getString("ra") +
                          ", dec: " + metadata.getString("dec"));
        }
     });
 }
```
Code example 32: The code snippet for accomplishing this scenario through using the MongoDB API driver

Scenario 3: Listing observation times for interesting images

Sometime we might find some images to be very interesting, for example if a cataclysmic variable occurred in them, and we want to know when these images have been captured. To list the observation time for these images, it is necessary to search the exposure metadata collection and use field, colour and run number as the query parameters. The Java codes for accomplishing this task are similar to those in the previous scenarios, so only the query syntax is presented in code example 33.

```
db.exposuremetadata.find({field: "gb5", colour: "R", chip: 0},{date:1})
```

Code example 33: The second parameter in the query syntax is used to limit the fields to return for all matching documents. In this example, the MongoDB query engine will only return the date field as the query output.

Scenario 4: Listing all metadata for exposures within a specified Julian Date range

In some analysis situations, the user might want to lookup exposure metadata which has been captured within a specified time range (JD), with the results sorted by the observation JD.

To achieve this goal, `$gt` and `$lt` operators provided by the MongoDB query engine can be used. The `$gt` operator will select those documents where the value of the field is greater than the specified value, and the `$lt` operator will select the value of the field in reverse order. The query syntaxes for accomplishing this scenario are as shown in code example 34:

```
db.exposuremetadata.find({ jdstart : { $gt : 2454580.07 },
                           jdend : { $lt : 2454606.9 } })
                           .sort({jdstart : 1})
```

Code example 34: The above query return all matching documents for which values of jdstart field are greater than 2454580.07, and values of jdend field are smaller than 2454606.90. The sort operator in MongoDB query syntax is used for specifying the field to sort by a particular sort order, which performs the same operation as the “orderby” command in the SQL context. For the second parameter of sort, 1 specifies ascending order, and -1 specifies descending order. In this example, we sort jdstart field in ascending order.

As can be seen, the query syntax for this scenario is a little bit more advanced than in the previous query and involves two-dimensional searches, but MongoDB has a built-in 2D search feature that is convenient for this type of search scenario. However, typing this query syntax in the command line could very easily make the mistake (i.e: missing a bracket). Thus, in order to avoid this mistake, we can utilise the MongoDB API driver to help us to construct the query syntax. An example code snippet as shown in code example 35:

```java
MongoCollection<Document> collection = getCollection("exposuremetadata");
//range1 specified jdstart great than 2454580.07
//range2 specified jdend less than 2454606.90
Document range1 = new Document("$gt", 2454580.07);
Document range2 = new Document("$lt", 2454606.90);
Document query = new Document ("jdstart", range1).append("jdend", range2);

//list all metadata that jdstart greater than 2454580.07
//and jdend less than 2454606.90 and Sorting query result
//in ascending order based on the jdstart field
FindIterable<Document> iterable = collection.find(doc).sort(new
                                  Document("jdstart", 1));
iterable.forEach(new Block<Document>()
{
  @Override
  public void apply(Document document)
  {
     System.out.println("run number: " + document.getString("run") +
                        " field: " + document.getString("field") +
                        " chip: " + document.getInteger("chip") +
                        " jdstart: " + document.getDouble("jdstart") +
                        " jdend: " + document.getDouble("jdend"));
   }
});
```

Code example 35: Java code for this scenario; three document objects need to be defined; the first one used for specifying the starting range, the second one used for specifying the ending range, and the third document object used for constructing the query syntax.

Scenario 5: Calculating RA and Dec from astrometric calibration coefficients

The astrometric calibration coefficients are used for calculating RA and Dec for transients. The entire procedure for this scenario is a little bit complicated and can be divided into three steps, as outlined in the points below:

- The program requires field, colour, chip, x and y as initial input parameters. x and y are pixel positions on the CCD chip, and these two parameters are used to determine the sub frame number (each CCD chip is divided into eight sub frames). For example, if input x=10 and y=10, the sub frame number will be 0.

- After the sub frame number is determined, field, colour, chip and sub frame number are used as query parameters for selecting corresponding astrometric calibration coefficient documents from the MongoDB database.

- Then the query results are retrieved from MongoDB and RA and Dec are calculated using a mathematical formula.

It must be noted that the processes for determining the sub frame number and mathematical formula for calculating RA and Dec are not the subject of this research. The MongoDB query syntax for this scenario are as shown in code example 36:

```
db.astrocoef.find({ field : "gb7", chip : 3, subframe : 0 })
```

Code example 36: The MongoDB query syntax for this scenario, similar to the previous scenarios, the MongoDB query syntax for dealing with this complicated scenario is still very straightforward and not much different with the previous scenarios.

The Java codes for lookup within the corresponding astrometric calibration coefficients document are not much different from the previous scenario, but it is still worthwhile to presents a Java codes example in order to demonstrate how to extract all of the coefficient data from the query results. A Java example is illustrated in code example 37:

```
MongoCollection<Document> collection = getCollection("astrocoef");
//assumes input of both x and y are 10, so the subframe number
//should be 0
Document query = new Document("field","gb7")
                     .append("chip", 5)
                     .append("subframe", 0);
FindIterable<Document> iterable = collection.find(doc);
iterable.forEach(new Block<Document>()
{
    @Override
    public void apply(Document results)
    {
      //Extract all coefficients data of selected document
      String ra = queryResult.getString("ra");
      String dec = queryResult.getString("dec");
      int sx = queryResult.getInteger("sx");
      int sy = queryResult.getInteger("sy");
      double x = queryResult.getDouble("x");
      double y = queryResult.getDouble("y");
      double mu = queryResult.getDouble("mu");
      double nu = queryResult.getDouble("nu");
      double phi = queryResult.getDouble("phi");
```

```
        double psi = queryResult.getDouble("psi");
        //Peforming RA and Dec operation here
    }});
```

Code example 37: The Java codes for extracting the astrometric calibration coefficient values from the query result.

After the above codes are run, the MongoDB API driver will search the MongoDB and return the selected astrometric calibration coefficients document. The mathematical formula codes for calculating RA and Dec for the transient object can be written within the `apply` method.

Scenario 6: Extract nearest star in R star catalogue

When an interesting transient object is found, we will also want to find all of the nearby catalogued stars for this interested object. This is a common search scenario in an astronomical database, and this type of search is also referred to as a two dimensional data search. The process for extracting the nearest catalogued stars is more complicated than all of the previous scenarios and the full procedure can be summarized as three steps:

- The program requires field, colour, chip, x, y and halfwidth as the input parameters. The halfwidth is a pixel value; the halfwidth is a constant value, and will normally be 10 pixels.

- As discussed in the previous section, a list of field name based collections (Figure 40) has been created to store the different field's stars catalog data. Therefore, when performing this query, the program will only search the nearest catalogued stars from the field collection that corresponds to the input field. For example, the gb1 collection will be accessed and searched if the input field is gb1.

- In fact, x and y values cannot be directly used in the query, because we want to list all stars that are within a search box about the input x and y values. The boundaries of the search box can be obtained through defining x ± halfwidth and the y ± halfwidth. In other words, the search range formula for this query is:

nearest_catalogued_stars = all stars in specified field-color-chip combination where x > x1 && x < x2 && y > y1 && y < y2.

In this query, the input x value is assumed to be 210, the y value is assumed to be 25, the halfwidth value is 10, and nearest catalogued stars that need to be extracted belong to the gb1 field. Code example 38 presents the query syntax of MongoDB for accomplishing this task.

```
db.gb1.find({ colour : "R", chip : 7, xcenter : { $gt : 200.0,
              $lt : 220.0 }, ycenter : { $gt : 15.0, $lt : 35.0 } })
```

Code example 38: In this query scenario, we know nearest catalogued stars that we want to extract are belong to gb1 field, and all stars documents of this field (gb1) are stored in gb1 collection. Thus, we only needs to lookup the gb1 collection to extracting all of the matching nearest catalogued star documents, without the need for lookup in another collections.

Through the above query example, it can be seen that MongoDB has the ability to deal with this complex two dimensional data lookup in one simple query command without using any other astronomical customized SQL search techniques (e.g. cone search [175]). In fact, constructing this query syntax through using a program is much easier and simpler than manually entering a query in the MongoDB command shell. The Java code snippet is as shown in code example 39:

```java
public void ExtractNearestCataloguedStars(String field, String colour,
                                          int chip, double xcen,
                                          double ycen, int halfw){
  MongoCollection<Document> collection = getCollection(field);
  //Defines search range for xcenter and ycenter
  double x1 = xcen - halfw;
  double x2 = xcen + halfw;
  double y1 = ycen - halfw;
  double y2 = ycen + halfw;
  Document xrange = new Document("$gt", x1).append("$lt", x2);
  Document yrange = new Document("$gt", y1).append("$lt", y2);
  Document query = new Document("field",field).append("colour", colour)
                      .append("chip", chip).append("xcenter", xrange)
                      .append("ycenter", yrange);
  FindIterable<Document> iterable = collection.find(query);
  iterable.forEach(new Block<Document>()
  {
      @Override
        public void apply(Document results)
      {
          //Display all nearest catalogued stars matching given conditions
          System.out.println("star id: " + document.getInteger("starid") +
                             " ra: " + document.getString("ra") +
                             " dec: " + document.getString("dec"));
      }});
}
```

Code example 39: Java codes for extracting all of the nearest catalogued stars for the same given parameters. As in the previous examples, the further analysis program logic can also be done within the `apply` method.

After the above code is run, the MongoDB will return all matching R star documents in the gb1 collection for which the value of the x field and the value of the y field are within the given condition range.

Scenario 7: Create Event Portfolio Document

This scenario involves creating an event portfolio document for each interesting transient object and storing them in a separate collection called an “event portfolio”. The flow chart diagram for creating an event portfolio document is illustrated in Figure 41 and five main methods for create an event portfolio document are shown in code example 40, with each method used to perform an individual task..

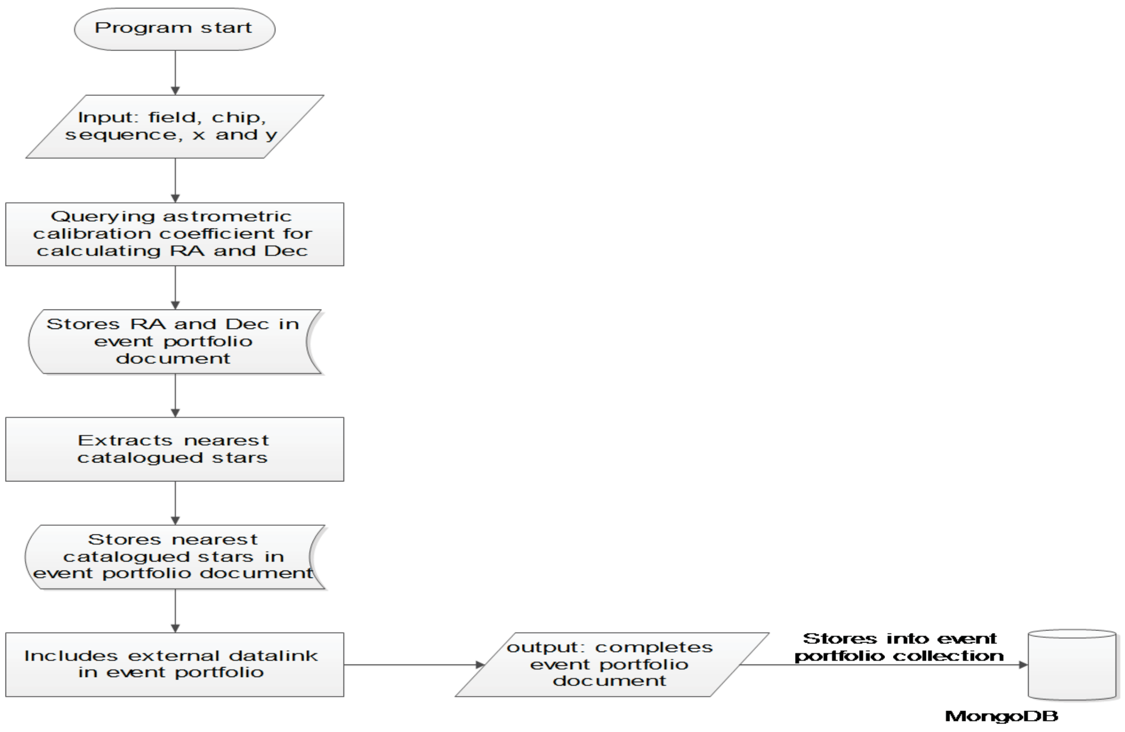

Program start
Input: field, chip, sequence, x and y
Querying astrometric calibration coefficient for calculating RA and Dec
Stores RA and Dec in event portfolio document
Extracts nearest catalogued stars
Stores nearest catalogued stars in event portfolio document
Includes external datalink in event portfolio
output: completes event portfolio document
Stores into event portfolio collection
MongoDB

Figure 41: Flow chart of creating an event portfolio document for an interesting object.

```
//Defines eventPortfolio as global variable
Document eventPortfolio = null;
public GeneratingEventPortfolioBSON (String field, int chip,
                                    double x, double y, int sequence){
   eventPortfolio = new Document();
   //creates internalID for object and stored in document
   String internalID = field + "-" + colour + "-" + chip + "-" + sequence;
   eventPortfolio.append("internalID", internalID);
   //Method used for stores all basic information of object in event
   //portfolio document
   IncludesBasicObjectInfo(field, colour, chip, x, y, sequence);
   //Method used for search observation date of specified run-field-
   //colour-chip combination from exposuremetadata collection, we assume
   //run number of this object is B41092
   GetObservationDate("B41092", field, colour, chip);
   //Method used for calculating RA and Dec
   CalculatingRAandDec(field, chip, x, y);
   //Method used for extracting nearest catalogued stars of object
   ExtractingCataloguedStars(field, colour, chip, x, y);
   //Method used for includes external data link in event portfolio
   AppendExternalDatalink(internalID);
}
```

Code example 40: Construction of event portfolio generator program.

First, the program will accept field, chip, sequence number, x and y (the x and y are pixel position on the CCD camera, there two parameters are also referred to ccdx and ccdy) as the input parameters. In this example, “gb10”, 3, 20370, 1263.61 and 1889.37 are used as the input parameters.

Second, use the methods discussed in two previous scenarios (scenario 5 and scenario 6) to calculate RA, Dec and to extract the nearest catalogued stars for the interesting transient object. Store the results in the event portfolio `Document` object.

Third, store the URL of externally referred data such as finder chart images (both FITS and JPEG formats), light curve plots, and photometry measurements data files. The file names of these external files are constructed following the same pattern (field-colour-chip-sequence.extension). For example, the file path of the finder chart image for the gb10-R-3-20370 object will be `$DATAHOME/finder-gb10-R-3-20370.jpeg` (JPEG format) and `$DATAHOME/finder-gb10-R-3-20370.fit.gz` (FITS format). So, it is only necessary to add a prefix in front of the file name pattern to construct the full file path for different types of files, as shown in code example 41:

```
private void AppendExternalDatalink(String internalID)
{
   //file path of external photometry data,finder chart images and
   //lightcurve plot image
   String externalPhotometry = "$DATAHOME/phot-" + internalID + ".dat";
   String finderChartImage = "$DATAHOME/finder-" + internalID + ".fit.gz";
   String finderChartImage2 = "$DATAHOME/finder-" + internalID + ".jpg";
   String lightCurveImage = "$DATAHOME/plot-" + internalID + ".png";

   //stores to event portfolio
   eventPortfolio.append("photometry", externalPhotometry);
   eventPortfolio.append("finderFITS", finderChartImage);
   eventPortfolio.append("finderJPEG", finderChartImage2);
   eventPortfolio.append("lightcurve", lightCurveImage);
}
```
Code example 41: Including links of all necessary external data files.

Finally, after all of the information has been gathered and stored in the event portfolio `Document` object, we can store this event portfolio `Document` object into the event portfolio collection. An example code for inserting new documents into a collection has already been presented in the previous section (Section 6.3.2), so this will not be discussed again here.

Moreover, the time consumed in creating and inserting event portfolios using both MongoDB and a relational database has been measured, with the measurement starting from the moment that that program is entered until the moment it completes inserting the event portfolio into the collection (or relational database row). The measurement results show that the MongoDB only takes 61 millionseconds to complete the whole process. Details of the performance analysis will be presented in the next section.

Scenario 8: Removing unwanted event portfolios

Not all interesting objects detected in the MOA data result in a formal event notification being sent out. Some may turn out to be spurious events caused, for example, by dust spots or bad columns on the CCD chip. Therefore, it will be desirable to remove these uninteresting event portfolio documents from the event portfolio collection in order to save disk space. For this scenario, only the internal id of the transient object is required as the input parameter (i.e. gb10-R-3-20370). In this scenario example, we will remove the event portfolio document created in the previous scenario. The MongoDB data removing syntax are shown in code example 42 below:

```
db.eventportfolio.remove({ internalID: ”gb10-R-3-20370” })
```

Code example 42: MongoDB syntax for removing an event portfolio based on internal ID.

As we can see, we need to specify the document ID to indicate which document needs to be removed from the collection, which is very similar to SQL. If one only wants to remove a single document from a collection, the MongoDB command shell can be quickly used to accomplish this task.

However, if it is necessary to remove multiple documents from a collection, it is better to use MongoDB API driver. Code example 43 illustrates this process using Java code to remove unwanted event portfolio documents.

```java
public void RemoveDocuments()
{
   //specified internal id of event portfolio documents
   //that need to removed
   String[] removeDocs = {"gb10-R-3-20370", "gb5-R-9-70270",
                          "gb2-R-8-9915"};
   Document removeDoc = new Document();
   MongoCollection<Document> collection = getCollection("eventportfolio");

    for (int i = 0; i < removeDocs.length; i++)
    {
      removeDoc.append("internalID", removeDocs[i]);
      //remove document from event portfolio collection
      collection.deleteOne(removeDoc);
   }
 }
```

Code example 43: Removing unwanted event portfolio documents from a collection with Java

After the above code is executed, the specified event portfolios will be removed from the event portfolio collection permanently.

6.5.2: Use Case of Alerting New Transient Event Notifications

As we discussed in chapter 3, the VOEvent packet is based on an XML format and it is a standard format for transient notification. In the previous system, the VOEvent packets are generated through selecting data from different data files, which requires a slightly complicated process. However, with MongoDB, the entire process for generating a VOEvent packet will be simplified, because an event portfolio document contains all necessary data for generating that particular VOEvent packet. This means that it is only necessary to retrieve the corresponding event portfolio document from MongoDB, and then use the data stored in the event portfolio document to generate the VOEvent packet. One can think of this VOEvent packet as representing a “snapshot” of the current state of the event in the database.

The full procedure for generating a VOEvent packet by using the event portfolio document can be summarized into the following steps:

First, the program only requires the internal id and message text as the input parameters; the message text is a short description about the new transient that needs to be alerted.

Second, a MOA-ID is assigned to the alerted transient object; the MOA-ID is a serial number that indicates the latest ID of the alerted transient object, and it will only be generated when a new transient event has been alerted. For example, if 100 transients have been alerted previously, then, the MOA-ID for the new transient event will be 101. An individual BSON document has been created for records of all the previous MOA-IDs, and this document is also stored in the event portfolio document. The structure of the MOA-ID records document is as in code example 44:

```
{
  filename: "moaidlist",
  MOAID: ["2015-BLG-098", "2015-BLG-099", "2015-BLG-100",
            "2015-BLG-101", "2015-BLG-102"]
}
```

Code example 44: BSON document structure of MOA-ID records. As can be seen, the BSON document supports array type as field value. In this example, all previous MOA-IDs are stored in an array as field value of MOAID field.

When we need to generate new MOA-ID, we will retrieve this document from the event portfolio collection, getting the last MOA-ID from the array (the last MOA-ID will be the last element of the array) and increasing the last MOA-ID by one to generate the new MOA-ID. The new MOA-ID generation operation is shown in code example 45:

```
1  private void createNewMOAID()
2  {
3    MongoCollection<Document> collection = getCollection("eventportfolio");
4    //Get last element from MOAID field through using "$slice" operator
5    Document query = new Document("MOAID", new Document("$slice", -1));
6
7    FindIterable<Document> iterable = collection.find().projection(query);
8    iterable.forEach(new Block<Document> ()
9    {
10       @Override
11       public void apply(Document results)
12       {
13           //get the lastest id from query result
14           lastID = results.get("MOAID").toString();
15           //because value of MOAID field is array, so the query result will
16           //display id within [] brackets. i.e:[2015-BLG-100], using
17           //substring method to removes brackets
18           lastID = lastID.substring(1, lastID.length() - 1);
19       }
20   });
21
22   String[] splitMOAID = lastID.split("-");
23   //increase MOA ID number by one
24   int idNumber = Integer.valueOf(splitMOAID[2]) + 1;
25   //construct new MOA ID
26   String newMOAID = splitMOAID[0] + "-" + splitMOAID[1] + "-" + idNumber;
27 }
```

Code example 45: Generating a new MOA-ID from the MOA-ID records document. The only way to lookup the last element within the array is to use a "slice" operator, which is a projection operator, and it is necessary to use this projection method to get the data from the array in MongoDB. The parameter value, -1, in line 5 specifies access to the last element of the array, and 1 specifies access to the first element of the array.

After the new MOA-ID has been generated, it needs to be appended into the MOA-ID records document for next time use. The document update code is as shown in code example 46:

```
private void updateMOAIDDoc(String newMOAID)
{
   MongoCollection<Document> collection =
                              getCollection("eventportfolio");

   Document find = new Document("filename", "moaidlist");
   Document push = new Document("$push",
                                 new Document("MOAID",newMOAID));
   UpdateResult ur = collection.updateOne(find, push);
    //if update operation successful, modified count will be 1
    System.out.println(ur.getModifiedCount());
}
```

Code example 46: Appending the new MOA-ID to the array in the MOA-ID records document.

It will also be necessary to update the new MOA-ID to the corresponding event portfolio document, because in the initial stage, all of the event portfolios will not have MOA-ID fields. In other words, only those event portfolio documents that have been used for alerting will contain a MOA-ID field. Therefore, it is necessary to create a new field in the event portfolio document and update the MOA-IDs to this new field, as shown in code example 47 below.

```java
private void updateEventPortfolio(String newMOAID, String eventMessage,
                                  String internalID)
{
   MongoCollection<Document> collection = getCollection("eventportfolio");
   //Creates MOA-ID field in specified event portfolio
   Document setFields = new Document("MOAID", newMOAID);
   //Create short description field
   setFields.append("eventdescription", eventMessage);
   //Create transient Identification field
   setFields.append("identification", "microlensing");
   //using set operator
   Document set = new Document("$set", setFields);
   //specified which event portfolio need to be updated
   Document find = new Document("internalID", internalID);
   UpdateResult ur = collection.updateOne(find, set);
   System.out.println(ur.getModifiedCount());
}
```

Code example 47: Creating and updating the MOA-ID, short description and transient identification fields in the event portfolio document

After all the above steps have been completed, it is possible to create a VOEvent packet using data stored in the event portfolio document, and then send the VOEvent packet to the dispatch program (refer to Chapter 4) for alerting new transient notifications.

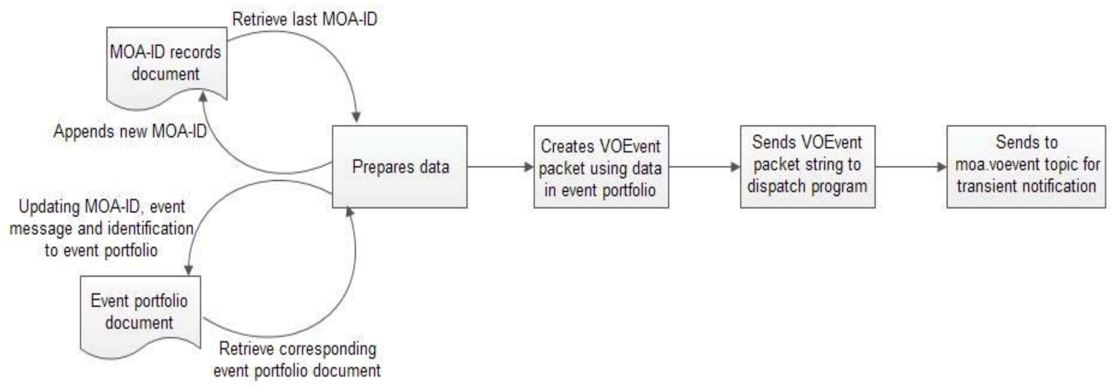

Retrieve last MOA-ID
MOA-ID records document
Appends new MOA-ID
Prepares data
Creates VOEvent packet using data in event portfolio
Sends VOEvent packet string to dispatch program
Sends to moa.voevent topic for transient notification
Updating MOA-ID, event message and identification to event portfolio
Event portfolio document
Retrieve corresponding event portfolio document

Figure 42: Work flow diagram for creating VOEvent packets using the event portfolio document.

6.6 Performance Measurements and Scalability Discussion

The data access performance is a very important factor for measuring the scalability of a database. A scalable database should provide fast data access ability for dealing with any volume of datasets, even when the volumes of data stored in the database are very massive. The LSST has much larger amounts of datasets that need to be stored in the database compared to MOA, thus, it is necessary to run this performance measurement experiment to determine whether or not MongoDB is able to be scaled up and used by LSST.

6.6.1 Performance Measurements of MongoDB

This section examines the data manipulation (read, write and delete) performance of the MongoDB database system, and the measurement tests are conducted under the following conditions:

- The MongoDB version used in this experiment is 3.0.1.
- The testing programs and MongoDB are run on the same machine. The machine has a hardware configurations of 2.75 GHz Intel i5-2500s CPU, 8GB Kingston DDR3 RAM, and runs a Windows 7 64bits operating system using the default installation settings.
- Two different types of data storage media are used in this performance experiment: SSD (Solid-State Drive) and spinning disk (HDD). This is because one of the factors that will impact the performance in accessing data (read/write) from database systems is the I/O throughput of storage media. As we know, SSD provides much faster and higher I/O throughputs than a traditional spinning disk. Thus, the experiment will also measure how much data access performance will be impacted by the data storage media.

The SSD used in this experiment is a Kingfast K9 with 128 GB capacity, and the read and write speed of this disk is 250 MB per second. Indeed, the SSD is the data storage media that is recommended by MongoDB.
The spinning disk used in this experiment is a Western Digital WD10EARX with 1 TB capacity, which has the ability to provide 64 MB read and write throughputs per second.

- Each measurement experiment will be run three times, and the results presented in the tables in this section are the mean values of the three time measurements.

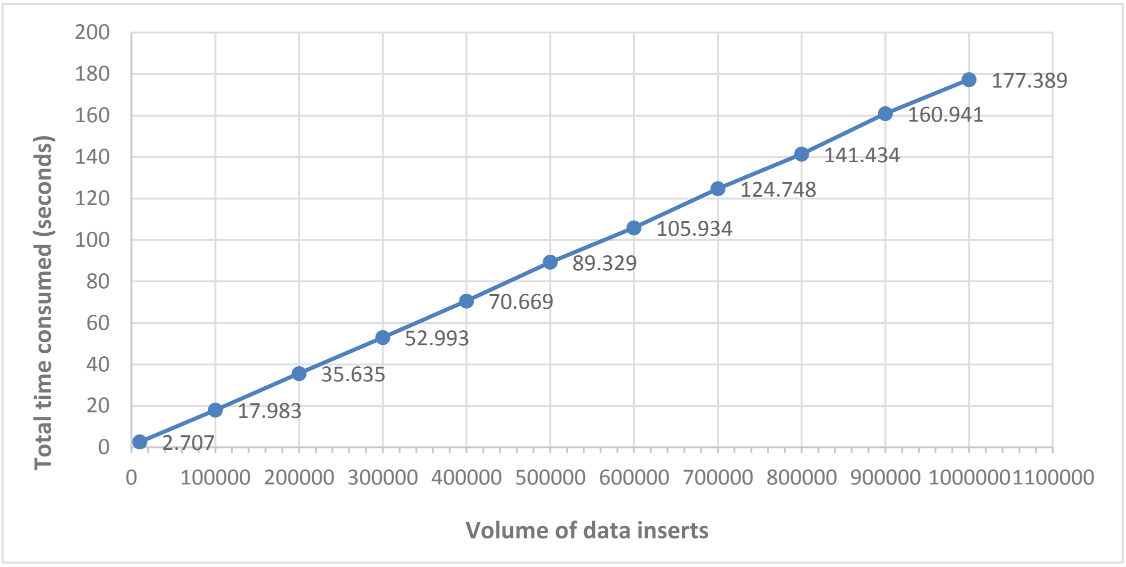
Performance of data insertion

Data insertion is the first step needed before manipulating the databases; without data insertion, it is not possible to perform any subsequent operations (read, write and delete). In the meantime, the performance of data insertion is also very important for all database projects, especially for big data generation projects such as LSST. For example, the LSST will generate approximately 300 MB of observation imaging data per second [176].

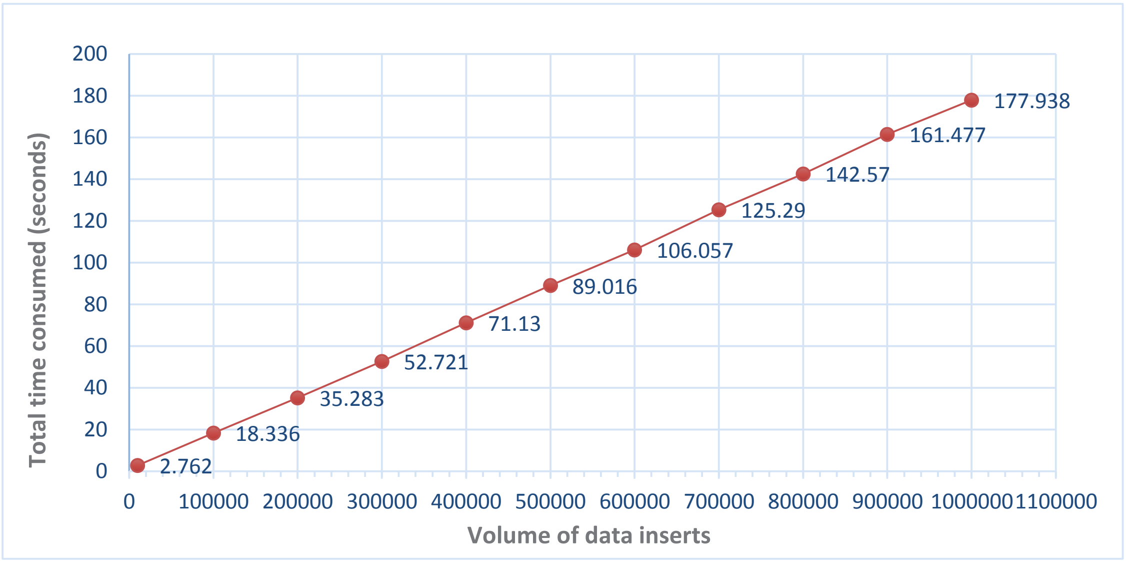
In fact, it is not desirable to archive all of the captured imaging into the database, but it is necessary to archive all of the scientific data (i.e. imaging metadata, measurement data, and so forth) into the database. The scientific data are smaller than the imaging data, so we are assuming that LSST would need to store 150 MB of data into the database every second. However, if the data insertion rate is slower than the data generation rate, then the data process system will generates lots of backlogged data, which means that users are not able to use the data instantly after it has been generated for real time analysis operations.

Therefore, the first step is to measure the data insertion performance of both databases before measuring performance of any other data manipulating operations.

In this measurement, different samples' data files will be used for insertion into the database, with the measurement of time consumed by the complete insertion of each sample in order to determine the data insertion performance. The size of each data file used for the inserts is 6 KB. Figure 43 and Table 8 show the measurement results for the inserts of the different samples of data into both databases.



(a)



(b)

Figure 43: Comparison of data insertion performance between an SSD disk and spinning disk. The top figure shows the performance from inserting data into MongoDB using a SSD disk as the storage media. The bottom figure shows the performance of inserting data into MongoDB using a spinning disk as the storage media. As we can see, MongoDB only requires less than three seconds to complete one million data insertion tasks, and the data insertion performance of using these two storage media are almost the same.

Samples of data inserts	Total time consumed of MongoDB (SSD)	Total time consumed of MongoDB (HDD)
10,000	2.707 seconds	2.762 seconds
10,000,0	17.983 seconds	18.336 seconds
20,000,0	35.635 seconds	35.283 seconds
30,000,0	52.993 seconds	52.721 seconds
40,000,0	70.669 seconds	71.130 seconds
50,000,0	89.329 seconds	89.016 seconds
60,000,0	105.934 seconds	106.057 seconds
70,000,0	124.748 seconds	125.290 seconds
80,000,0	141.434 seconds	142.570 seconds
90,000,0	160.941 seconds	161.477 seconds
10,000,00	177.389 seconds	177.938 seconds

Table 8: Time consumed for inserting each sample of data with different types of data storage media.

Based on the measurement results shown in Figure 43 and Table 8, it can be seen that the time consumed for inserting the data into MongoDB increases linearly, with the time variation caused by the system schedule and disk I/O (input/output). The time consumed in using an SSD as the data storage media is almost the same as using a spinning disk as the data storage media, which means that the I/O throughput of the data storage media will not make any impact on the data access performance.

However, there is one interesting point presented in the MongoDB measurement results, which is that SSD does not really speed up the performance over that of spinning disk, and the time consumed of using SSD as the data storage media is almost same as using spinning disk as the data storage media. The main reason for this is that MongoDB uses memory mapped files for storing and interacting with all of its data. All MongoDB data files are mapped to the virtual memory region using `mmap()` primitive [177], which is a similar concept to pointer. So, each data file in the disk block will have a corresponding address in the virtual memory, and MongoDB can read or write the contents of its data files as if they were in RAM. In other words, data read and write operations in MongoDB are performed in RAM. As is known, manipulating data in RAM is much faster than manipulating data from disk (DDR 3 1333 RAM can provide approximately 10 GB read and write per second).

MongoDB will only flush all data stored in the RAM to disk at a predefined interval (this interval can be configured through setting `–syncdelay` parameters at the start up of MongoDB instance, and the flush interval by default is 30 seconds). This flush operation is done in the background asynchronously. In other words, data insertion performance is dependent on the IO throughput of the RAM, not the disk; so the disk I/O throughput will not have much affect on the data insertion performance. As mentioned at the start of this chapter, all measurements are done under the same RAM. This is why the SSD does not speedup data insertion performance over the HDD. It is pleasing to note that using a faster RAM (i.e. DDR3 1600) can obtain better data insertion results than the measurement results obtained in this experiment.

In fact, mapping data files into the virtual memory region does not mean that all these data files necessarily have to be loaded into the physical RAM as well. MongoDB will only map those data files that need to be accessed to the physical RAM, with all unaccessed data files only mapped in the virtual memory region. Figure 44 shows the internal work flow of the MongoDB database.

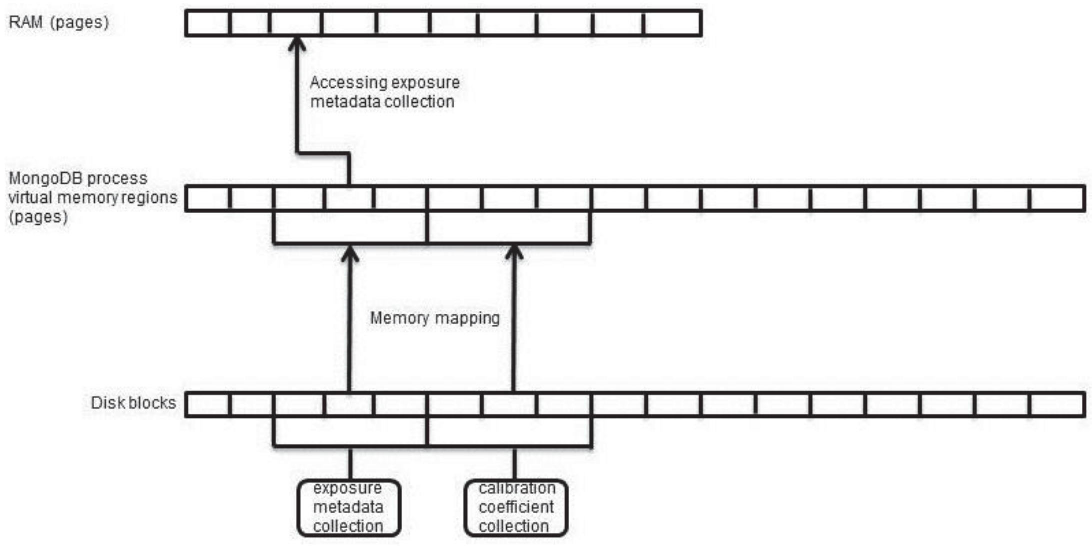

RAM (pages)
Accessing exposure metadata collection
MongoDB process virtual memory regions (pages)
Memory mapping
Disk blocks
exposure metadata collection
calibration coefficient collection

Figure 44: Workflow of internal data interaction for accessing data files. As can be seen, all data files (collections and documents) under the same database namespace are stored in contiguous disk block regions.

When data (documents) are inserted into the MongoDB, the page cache will check if the page of the data file that needs to have the data written into it exists or not. If the page of a data file does not exist, then a page fault occurs, and the OS kernel will tell the page cache to find an available page to load the required data file from the disk, map it, and then immediately performs data written operation.

Once the data has been successfully written to the relevant page in the RAM, the MongoDB will perform subsequent data writing immediately. In addition, performing real time analysis operations becomes possible with MongoDB, because once data are available in the RAM, they can be immediately accessed by other programs.

Moreover, from table 7, the total amounts of data insertion that can be handled by the database in each second can be calculated using a simple formula; total volumes of data inserts/total time consumed. The calculation results show that the MongoDB database can handle approximately 3000 data insertions in each second, which means that approximately 180 MB (data size for each insert is 6KB) of data can be inserted into MongoDB each second.

Based on measurement results and the above discussion, MongoDB has ability to provide extremely fast data insertion speed, and this data insertion performance fully satisfies the LSST requirements.

Performance of data querying

Data explosion performance is also very important in astronomical database systems. This is because we use data retrieved from database systems to perform some real time analysis operations. Therefore, a suitable database system must have the ability to respond to query results within a very short timeframe, even when large amounts of datasets are stored in the database system.

The dataset sizes used for this measurement are the same as for the data insertion measurement task, and we will extracts the nearest catalogued stars from given combinations (field, chip, x, y and halfwidth) in order to determine the query performance of MongoDB database. The values of the query input combinations for this measurement are; *field=gb1*, *chip=7*, *x=630.33*, *y=380.88*, and *halfwidth=10*, and these values are used throughout this data querying performance measurement.

In addition, through the previous measurement, it has already been found and discussed that the data storage media will not impacts the overall performance of MongoDB, so we will only measure query performance with SSD data storage media. The measurement results are shown in Figure 45:

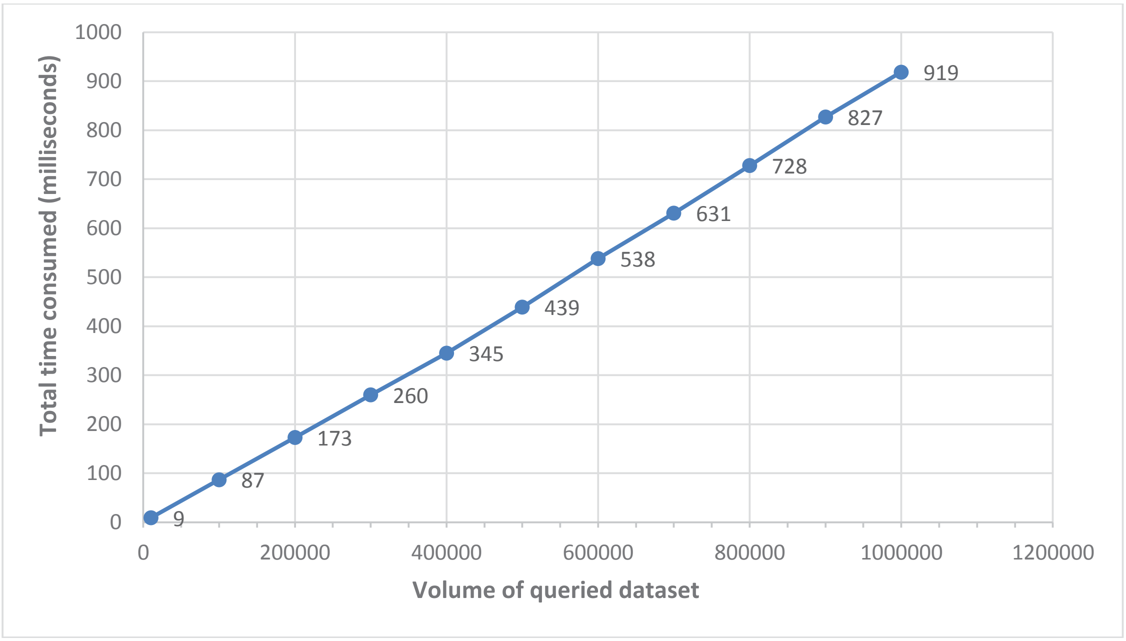


Figure 45: Data querying performance of MongoDB with different samples of datasets

Based on the measurement results, when the volumes of data increase, MongoDB requires more time to look up and extract all of the matching nearest catalogued stars from the databases, and the total time consumed for querying increase linearly.

The measurement results show that the MongoDB only takes less than a second to locate and retrieve all of the matching documents from the dataset that contains 1 million pieces of data. The main reason for this fast query performance is that all of the read and write operations in MongoDB are done in RAM, as previously mentioned. When performing the query operation, MongoDB will look up data from the corresponding pages in the RAM instead of on the disk, which is why the query performance of MongoDB is extremely fast.

Note that one thing we should be aware of is that MongoDB suggests that a working dataset should be able to fit in the RAM, so MongoDB can serves all queries from memory. The working dataset is not whole database; it is the particular collection that is needed for queries. In our example, the working dataset is a gb1 collection, and the sizes of all collections in our database system do not exceed 3GB, which could fully fit in the RAM of our testing machine. However, if the working dataset is much larger than the RAM, the cache misses will occur on a regular basis, and MongoDB will query the data from the disk. Thus, MongoDB also suggests using SSD as the storage media in case the working dataset does not fit in the RAM, because the IO throughput of an SSD is much faster than that of a HDD disk; so even if the working dataset cannot fit in the RAM, the MongoDB is still able to offer efficient query performance (of course, this will not be as efficient as querying data from the RAM). In other words, deployment with enough memory to fit the working dataset in the RAM will be able to achieve the best query performance.

In fact, in a real situation, no one would want to fit a large sized working dataset in the RAM (even when there is enough RAM to fit the dataset in). If a single collection contains a large sized dataset, then it is better to divide this large dataset into several subsets in order to achieve the best query performance and avoid many cache misses.

Based on the measurement results, the query performance of MongoDB is very efficient, and it also presents a good query performance while searching the data from large volumes of dataset. This query performance fully satisfies the MOA data rates, and should also be good enough for the LSST data rates.

However, one question that has always been asked is: is MongoDB still able to offer fast query performance when facing 10 million, 100 billion, or even larger data items situations?

In fact, there are two strategies that can be utilized to ensure query performance in massive dataset situations. The first strategy is to reduce the amount of documents in each collection, which means that the entire dataset is divided into several collections, so that each collection only contains a subset of the whole dataset. For example, if a particular field and chip combination contains 10 million star data documents, these star data documents can be divided into five collections, so that each collection only contains 2 million star data documents, with the nearest catalogued stars only needing to be extracted from the appropriate subset collection instead of extracting the nearest catalogued stars from whole dataset.

As discussed earlier, MongoDB is based on a dynamic schema, meaning that adding new collections can be done in real time without impacting on other existing collections and documents. Therefore, a new collection can be created at any time as needed. This is a simple and easy way to ensure query performance in massive dataset situations.

Another strategy is a little bit more complicated, but much more scalable than the first strategy; this is the use of the data sharding [178] feature offered by MongoDB. The basic idea of the data sharding feature is partitioning an entire dataset over multiple servers (shards) rather than storing all of the data in a single server, and each shard only contains a subset of the whole dataset. When the size of the dataset is increased, more servers can simply be added to support data growth and guarantee query performance.

Performance of data deletion

The performance of data deletion is not quite so important compared to data insertion and data query operations, because the deletion operation can be run in the background and other operations can be performed in parallel.

In fact, deleting data from a database is a very rare operation, as all historical records should be kept in the database for tracking purposes. However, it is still worthwhile to measure the performance of data deletion, because a lot of system resources will be consumed if a single data deletion operation takes a very long time to be performed.

The sizes of the datasets used in this measurement are 10 thousand and 100 thousand, because it would be very rare to remove more than 100 thousand pieces of data from a database at the same time. The measurement results are presented in Figure 46:

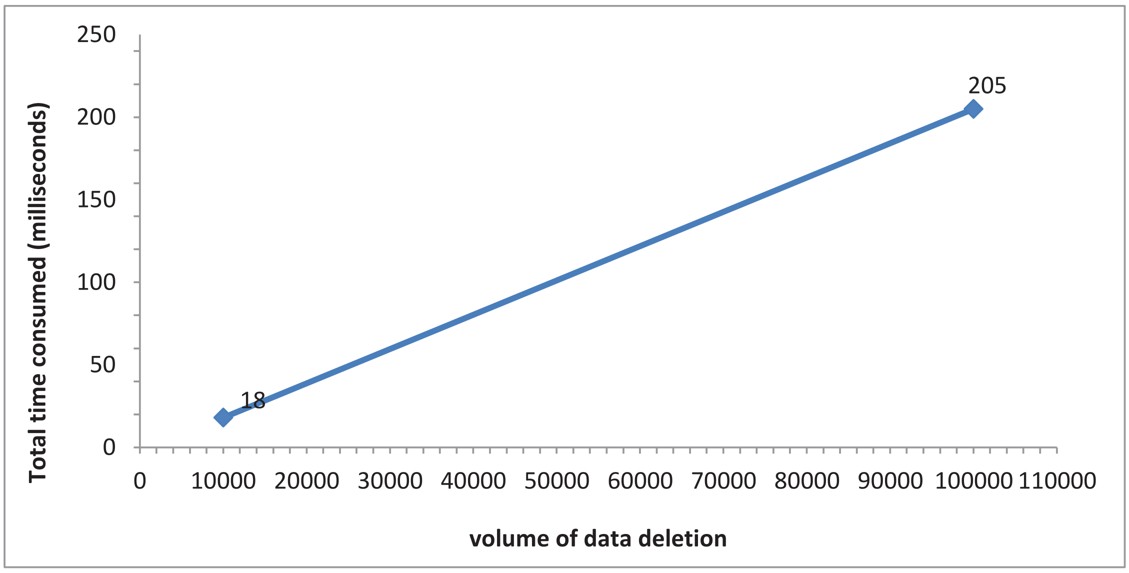


Figure 46: Data deletion performance of MongoDB.

According to the measurement results, removing data from MongoDB is also very fast and this result is as expected. However, as we can see, the time consumed for removing data from MongoDB is bit longer than for a query operation (with the same sized dataset), because when removing data from a database, the database will first locate all records that match the deletion criteria, and then perform the data deletion operation. The data deletion operation can be treated as if it were a combination operation (query and delete) and, therefore, a deleting data operation is always longer than a query operation.

Note that when deleting data from MongoDB, delete changes will not write to the disk instantly. The delete changes are only made in the RAM and then later synchronized to the disk when the flush interval has passed.

6.6.2 Additional Discussion of MongoDB Scalability

The performance of data insertion and data querying are not the only factors in estimating the scalabilities of database systems, because in large database systems, data growth is very fast (i.e. LSST, LinkedIn, and Facebook) and users may query data or read data from the database at a very high frequency rate. This high frequency I/O throughput and the large size of the datasets can challenge the capacity of the server hosting the database system. Therefore, database systems must be capable of meeting the demands of these increased workloads.

Basically, database systems utilize two approaches to address these increased workloads. The first approach is called “vertical scaling” [179] (also referred to “scaling-up”), and this scaling approach is mainly used by relational database systems. With vertical scaling, it is possible to increase capacity simply by adding more powerful CPUs, larger amounts of RAM and larger capacity storage resources to a server in order to deal with large workload situations.

However, the vertical scaling approach has two general issues. First, vertical scaling usually requires downtime while new hardware is being added, which means that the whole database system needs to be taken offline. This action is unacceptable for any large database systems. For example, when LSST begins operating, they might want their system to run 7 days a week and 24 hours a day in order to keep providing services to the users (public and internal) and write the observation data into the database without any downtime. Thus, the vertical scaling approach is not suitable for LSST.

Second, adding large amounts of powerful hardware resources can be very expensive, and each server has hardware limits. Once the hardware limits has been reached, there is no way to increase the capacity of the server in order to handle much larger I/O throughput and store more data.

Another scaling approach is called “horizontal scaling” or shards [180] (also referred to as “scaling out”). With this approach, the data are divided over multiple servers instead of storing the whole dataset in a single server. In other words, it is possible to increase capacity by connecting multiple servers together with all of these servers working as a single logical unit and, therefore, all bottlenecks that exist in vertical scaling can be resolved.

The MongoDB and NOSQL databases mainly use this scaling approach to handle increasing workloads. Indeed, relational databases can also scaled up horizontally, but the processes to do so tend to be complex and expensive, and it is easy for errors to occur in the process [272][273][274][275][276].

The horizontal scaling approach has a lot of advantages over the vertical scaling approach, with the major advantages described as follow:

- The horizontal scaling approach is much more economic than the vertical scaling approach, because it is not necessary to purchase and install a lots of powerful hardware resources into a single server. The whole dataset can simply be divided into multiple small servers, or inexpensive commodity computers, to reach the same capacity as a powerful server.

- It is possible to add and remove servers from the server cluster dynamically, which means that the storage and I/O throughput capacity can be increased on the fly without taking the whole system offline.

- The horizontal scaling approach reduces the volume of I/O throughput and data storage that each server needs to handle, while cluster growth (lots of servers) means that each server needs to process fewer I/O throughputs and store less data. In other words, the workloads of each single server will be reduced significantly.

As mentioned in the previous sub section, MongoDB offers a feature called data sharding to support horizontal scaling. The sharding cluster architecture is illustrated in Figure 47:

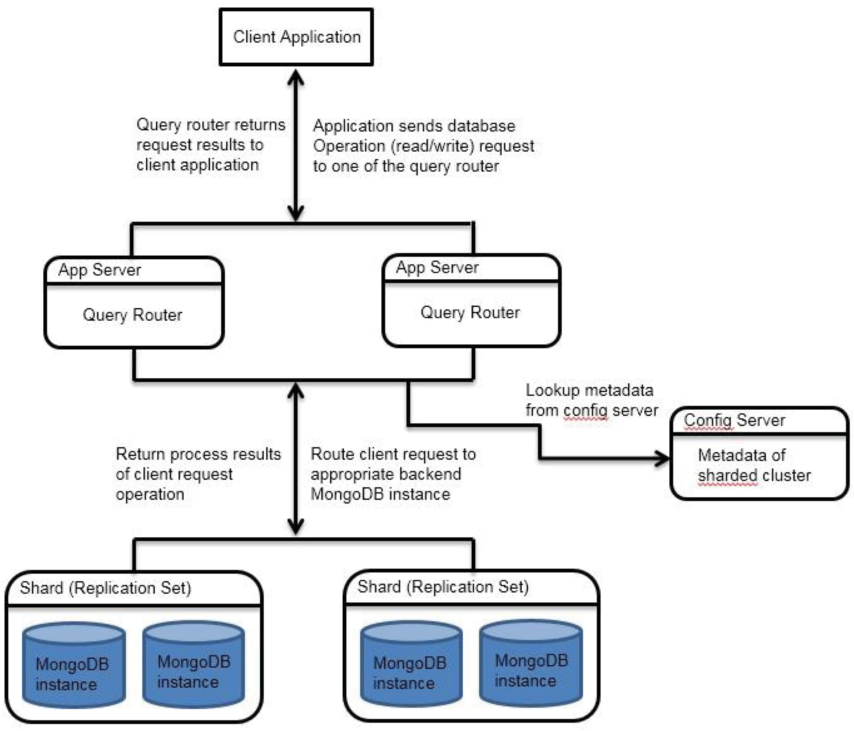

Client Application
Query router returns request results to client application
Application sends database Operation (read/write) request to one of the query router
App Server
Query Router
App Server
Query Router
Lookup metadata from config server
Config Server
Metadata of sharded cluster
Return process results of client request operation
Route client request to appropriate backend MongoDB instance
Shard (Replication Set)
MongoDB instance
MongoDB instance
Shard (Replication Set)
MongoDB instance
MongoDB instance

Figure 47: MongoDB sharding cluster architecture. All of the databases operations requests must go through query routers.

Figure 47 shows that a MongoDB sharding cluster consists of three major components; shards, config servers, and query routers. The role of each component is briefly described below:

- **Shards**: The shards are used for storing actual data (collections and documents). Each shard is a replication set and contains one or more MongoDB database instances, with the replication set providing automatic failover and redundancy, which can guarantee availability and ensure data exists on any number of servers across multiple data centres. Therefore, even if a failure occurs in a single server, the applications are still able to look up data from backup servers.

- **Config servers**: The config servers are a special Mongo instance and contain all of the metadata of the sharding cluster. The metadata contains the dataset details of each shard, such as what collections and documents are held by each shard.

- **Query router**: The query router is the access interface with the client applications. It routes all of the application requests to the appropriate backend shard, or shards. In other words, in a sharding cluster environment, the applications are connecting to a query router, rather than directly connecting to a MongoDB instance.
  When a query router has received operation requests (read or write) from an application, it will first look up the metadata from the config server in order to determine which shard contains the requested data, and then route the request to the corresponding shard server. Whenever the shard server completes the request operation, it will return the process results to the query router, and the query router will return the results to the client applications. MongoDB suggests that each sharding cluster should contain more than one query router in order to disperse application request loads.

An unlimited number of shards can be added into a sharding cluster, so it is possible to just add more shards to handle massive datasets and I/O throughputs. In addition, when enabling the sharding feature, the sharding cluster will automatically balance the whole dataset into the different shards. For example, if the whole dataset is 10 TB and the sharding cluster contains 10 shards, each shard will only be assigned 1 TB of data.

Note that data sharding is not implemented in this research, because a single server is adequate to deal with the MOA workload. However, the steps for setting up a sharding cluster in MongoDB are very easy; MongoDB provides full details of how to enable the data sharding feature on their official website [181].

Furthermore, LSST should definitely consider using a horizontal scaling approach to handle their large datasets and high frequency I/O throughputs instead of using a vertical scaling approach, and MongoDB is capable of handling the workloads that will be involved in LSST.

6.7 Conclusions

In this chapter, we have introduced the concept of using a NoSQL database for storage and dealing with astronomical data. In fact, the NoSQL database is a relatively new technology, and it has been designed as an alternative data management solution to the RDBMS system, rather than designed to replace the traditional RDBMS system. This is because the RDBMS system has existed for a long time and everyone is familiar with SQL and RDBMS, and it still does a good job on data management.

In fact, the NoSQL database is a very good data management solution and provides good scalability when facing big data situations. It has been found that MongoDB is very helpful for handling the dataset in this research, and that BSON/JSON documents are very good data representation formats for data organization. Moreover, based on the measurement results, the MongoDB shows good performance while dealing with large quantity datasets situation. Thus, MongoDB could be an ideal solution for storing and managing astronomical data, and it should present no problems to being scaled up to meet the demands of LSST data rate situations.

It is worthwhile considering NOSQL for next generation astronomical projects such as LSST that will generate large data volumes. There are many NOSQL database systems available, such as BigTable, CouchDB, Cassandra, and many others, but MongoDB is one possible data management solution that is worthwhile for consider by LSST.

Chapter 7
Conclusions and Future Developments

Real time transient notification and transient data management are the main computational challenges in all astronomical transient research fields. In this research, we have developed solutions to resolve both real time transient notification and transient data management challenges for MOA. These solutions can be used by any other astronomical transient projects for resolving the same challenges.

In fact, there are lots of new generation astronomical transient projects under development, or already in operation. These new generation transient projects will alert a lot more transient events and generate large amounts of observation data compared to the MOA project. Thus, there are great opportunities to extend this research to create a greater contribution to the field. This chapter will discuss some ideas for future development.

7.1 Future Development for Real Time Notification Architecture

We have developed a JMS based architecture for transient notification challenges, providing a very scalable solution to this problem. The JMS based transient notification architecture has the ability to alert a large amount of transient events at high event rates, and the JMS supports various messaging formats, which allows the message sender to send any type of data to the receivers. Additionally, the JMS based architecture is much more stable than the XMPP based architecture under the Pub/Sub mode.

In our case study, users who have subscribed to the "`moa.voevent`" topic will receive all VOEvent packets that are alerted by MOA, but this policy would need to be modified when facing large data rate situations. For example, the LSST may alert huge amounts of different types of transient events in a very short period. In this situation, using the same policy as in our case study would not be suitable, because this will generate lots of extra network bandwidth on the user side. Moreover, at most times, users might only want to receive some particular type of transient notifications rather than all of them.

In order to resolve this large data rate transient notification issue, a filter feature could be employed to select those events that are to be stored (i.e. implemented as a custom XPath, or XQuery string). The filter feature could be implemented at either the server side, or the application side, or possibly both. A server side filter will reduce the network traffic because when a new transient event needed to be alerted, the server will read a file that has recorded a custom query string and only push those events that match the users' filter criteria. Note that SkyAlert already implements this custom filter feature on their notification system. Users are allowed to build their custom filter criteria when they register on SkyAlert. Afterwards, users will only receive those event notifications that match filter criteria they have provided [15].

Furthermore, a custom topic for each user could also be established on the MOM broker, which is subscribed to when running the subscriber application (the subscriber application could be either desktop applications, or mobile applications). With custom topics, each user will only receive those transient notifications that match their interests.

7.2 Future Work for the Mobile Application

The primary goal of the mobile application study was to demonstrate how to utilize a smart mobile application to receive notifications of astrophysical transients by communicating with an MOM broker via the STOMP protocol. The first version of our application contains just a basic set of user features designed around our own preferences as users of our application.

However, there are many other possible features that can be added to future versions of this application, and a non-exhaustive list is as follows:

- Event stream selection. Currently our application only subscribes to the topic set up on ActiveMQ for MOA events. However, an MOM broker could provide a number of pub/sub topics for different event streams for various telescope projects. In future versions, we plan to allow the ability to subscribe to multiple topics together with an appropriate UI (probably implemented as checkboxes).

- Currently the events are all displayed in a scrollable list. A search feature could be added to allow just a subset of events to be displayed.

- There is no zoom or pan feature when displaying individual events. These features could be added.

- User definable reconnection period. As described in the previous section, during connection failure, the application will attempt to reconnect with the ActiveMQ server at a fixed time interval (30 seconds). This time could be set by the user instead.

- Automatic removal of unwanted events. Current users are required to manually delete unwanted events from the device storage. Users could be provided with the option of offloading stored events to a remote backup device, such as a desktop PC.

- Currently the device vibrates for every event received, but does not emit any sound. We plan to include user selectable options for vibrating and sounds in response to incoming events. Clearly, one would want the option of switching off the vibrate feature in the case of high event rates. Another possible feature could be to allow the user to attach different ringtones for different classes of events.

7.3 Future Development for the Astronomical Database

There are many different types of data that will be captured and generated during observation, so how to manage these data in an efficient manner is another important computational challenge. Using RDBMS systems for data storage and management is one possible solution, but RDBMS systems require predefined schema, which means modifying the database structure to meet future changes can be a very complicated and time consuming task. Moreover, the performance results outlined in Chapter 6 show that RDBMS systems encounter performance bottlenecks when handling large volume dataset situations. Thus, the RDBMS system is not a suitable solution for astronomical data storage and management.

The NoSQL database is a much more flexible, cost effective and scalable data management solution when compared to RDBMS systems, so the concept of utilising a document based NoSQL database (MongoDB) for storing astronomical data has been developed. The MongoDB is a document based NoSQL database; with this type of database, it is possible to add new collections for storing any new types of data, or modify the document structure in real time without affecting any other data stored in the database. In addition, the MongoDB database shows impressive data manipulation performance when facing a massive dataset situation and handling large workloads with MongoDB is also very easy.

In theory, any type of data can be formed into a document format and stored in the document-based NoSQL database. Therefore, all of the external photometry data, calibration data, observation log files, and even observation imaging (converted to binary format) can be stored into the database in the future. Another development to be considered is combining the database and mobile application together so that users can access and query data on their mobile devices. This is a very effective method for increasing collaboration between astronomers and survey groups.

7.4 Concept of using the JSON Format for Transient Notification

The VOEvent packet is based on an XML format; XML requires open and close tags to make this format work properly. Furthermore, the data structure of the XML format is verbose, redundant, and hard for both humans and machines to read, as discussed in Chapter 6.

JSON is another data representation format that is growing in popularity, it is smaller, compact and faster data exchange format compared to XML. Unlike XML, the JSON format does not need any open and close tags, and the data structure of JSON maps more directly onto the data structure used in modern programming languages, such as Java, Python, Javascript and among others. The programs can very easily access any elements in a JSON document through the specified field name. Moreover, all data carried by XML can also be carried by the JSON format. Thus, it is worth considering using JSON as an alternative transient event notification format. A possible JSON data structure for carrying notification data is illustrated in Figure 48 below:

```
{
  description: "MOA Transient Alert System",
  who: {
         shortName: "MOA", contactName: "Ian Bond",
         contactEmail: "i.a.bond@massey.ac.nz",
         Date: "2015-02-10T23:24:54"
       },
  wherewhen: {
               observatoryLocation: "MOA",
               AstroCoordSystem: "UTC-ICRS-TOPO", Time: 2457067.157764,
               Position2D: {unit: "deg", C1: 268.573115673,
                                 C2: -29.0070496334, Error2Radius: 0.0}
             },
  moa_id: "2015-BLG-003",
  event_description: "A transient event found by MOA",
  reference: "https://.../display.php?id=gb5-R-9-87873",
  discovery_image: {
                     domain: “BLG”, run: "B48266", field: "gb5",
                     colour: "R", chip: 9, seq: 87873,
                     ccdx: 1742.7, ccdy: 3781.62
                   },
  nearest_catalogued_stars:[
                           {
                             star_id: 161347,
                             ra: 268.57365563,
                             dec: -29.00731171,
                             magnitude_I: 17.528,
                             error_I: 0.140,
                             distance_to_event: 2.392659
                           },
                           {
                             star_id: 161418,
                             ra: 268.57310467,
                             dec: -29.00835896,
                             magnitude_I: 16.128,
                             error_I: 0.115,
                             distance_to_event: 4.302320
                           },
                           {
                             star_id: 161445,
                             ra: 268.57279025,
                             dec: -29.00642721,
                             magnitude_I: 15.949,
                             error_I: 0.066,
                             distance_to_event: 2.959347
                           },
                           {
                             star_id: 161550,
                             ra: 268.57187244,
                             dec: -29.00687869,
                             magnitude_I: 16.159,
                             error_I: 0.056,
                             distance_to_event: 3.649626
                           },
                            {
                              star_id: 161551,
                              ra: 268.57183893,
                              dec: -29.00756616,
                              magnitude_I: 16.301,
                              error_I: 0.118,
                              distance_to_event: 3.781738
                            },
                           ],
photometric_calibration: {
                             zero_mag: 27.3454690076,
                             zero_err: 0.350915610873
                           },
paczynski_model: {
                   t0: {value: 2457107.57958,
                        error: 23.3460327779, units: "JD"},
                   te: {value: 107.512384682,
                        error: 40.8015873215, units: "d"},
                   u0: {value: 5.68440731098e-05,
                        error: 0.10324845945, units: "-"},
                   Ibase: {value: 17.592211523,
                           error: 0.350915610873, units: "mag"}
                  },
external_photometry:{
                       reference: "https://.../phot-gb5-R-9-87873.da
                       reference: "https://.../plot-gb5-R-9-
87873.png"
                      },
finder_charts:{
                 reference: "https://.../finder-gb5-R-9-
87873.fit.gz",
                 reference: "https://.../finder-gb5-R-9-87873.jpg"
               },
importance: 0.5,
inference: {probability: 1.0, concept: "microlensing"}
}
```

Figure 48: Holding transient notification data through using the JSON format.

As can be seen, using the JSON format for carrying and presenting transient notification data is much more simple and straightforward than the XML format. In fact, all existing astronomical software (data analysis, robotic telescope and others) are highly customized for reading and analysing VOEvent packets. Thus, using JSON as the notification format will require lots of future development and software codes to be changed, so it still has a long way to go, but it is very worthwhile to purpose this option, and maybe some projects could implement this concept in a real environment in the future.

7.5 Conclusions

This thesis is focused on researching a possible solution for disseminating astronomical transient event notifications and studying the feasibility of using a NOSQL database for astronomical data management. The research makes three major contributions to astronomical domains, as outlined below:

1. Developing the concept of using JMS based architecture for astronomical transient event data delivery. This architecture has the ability to deliver transient event data to the intended recipients within a very short time. Since all the existing astronomical transient event notification systems are based on XMPP, this JMS based architecture could offer an alternative solution for future surveys that need to implement similar event notification systems. Additionally, the concept of this JMS architecture is not customised for delivering astronomical data; it could be adapted to other domains that require the implementation of a system for delivering notifications to the users.

2. A new method of using a mobile application to receive messaging notifications from message oriented middleware via the STOMP protocol was introduced. The goal of this study is to provide an alternative messaging notification solution on a mobile platform instead of using the default notification service offered by the mobile platform providers. We have developed an Android application for

demonstration purposes, but this theory is not limited to the Android platform, because the STOMP protocol offers a wide programming language library, so it is possible to develop an application on other platform and use the same method as we propose in this thesis to access the JMS message oriented middleware.

3. Developing a new concept of NOSQL database for astronomical data management, and presenting empirical works of how to map the astronomical data into the NOSQL database, as well as providing examples of querying astronomical data from the NOSQL database based on real astronomical scenarios.

The transient event notification architecture and Android event receiver application discussed in this thesis have been deployed since 2012, with some of the users starting to write their own receiver application or use Android application for receiving transient event notification from MOA. On the other hand, since the NOSQL solution for astronomical data management is a very new concept in astronomical domain. Therefore, this solution is only used by MOA for archive their observation data in current stage. In fact, the movement from one system to another system in the astronomy domain is slow, but it is expected that more users will come to use this architecture and NOSQL concept for manage astronomical data in the future.

Chapter 4 describes the implementation details of the JMS based transient event notification architecture and introduces how to disseminate and receive transient event notifications within this architecture. JMS is one of instant messaging technique which offers similar features as XMPP. However, JMS provides three attractive features: it supports wide range of message formats; it is possible to transport most of type data over the network via using JMS API. Second, JMS uses message objects for carrying and transporting the messages, the measurement results in this chapter show JMS is capable to transport large size of packets with a high efficiency. Last, JMS is also a interoperability messaging solution, the applications implemented with different JMS providers are able to communicate with each other. For example: a client application using ActiveMQ as the JMS provider can communicate with another client application using the SonicMQ provider. Moreover, JMS clients can interoperate fully with non-JMS clients; this is very important feature for distributed architecture environment. We have discussed how to access JMS based message oriented middleware from another programming language via the STOMP protocol within this chapter in order to demonstrate the interoperability of JMS.

A set of benchmarks have also has been conducted to examine the performance of this JMS based architecture. The results show that this architecture has the ability to publish and receive large amounts of packets within a short time period, and the overall performance is capable of meeting the demand of a large event rate situation.

Chapter 5 extends the study from Chapter 4 and mainly focuses on introducing the concept of receiving transient event notifications from message oriented middleware on a mobile application via the STOMP protocol. An android application has been developed for receiving transient event notifications from our message oriented middleware, and the fundamental framework design of this application is also discussed in detail within this chapter. This mobile application offers astronomers the ability to quickly receive, view and access the profiles of transient events on their mobile device. The performance benchmark of this mobile application is performed under a 3G network environment. According to the results, the performance of the mobile application is slower than on a desktop based application, but is still good enough to deal with the event rates of existing astronomical projects. However, if the same experiment is performed under 4G or faster network environments, it is expected that the results will be faster than the results presented in this thesis.

In Chapter 6, the focus of the study turns from transient event notification architecture to a NOSQL solution for astronomical data management. The goal of this chapter is to investigate the feasibility of a NOSQL solution for managing astronomical data. It is found that document oriented stores databases are ideally suited to this purpose. Thus, the MongoDB is used as a NOSQL database product for this study, with the full design of using MongoDB to manage astronomical data discussed in detail within this chapter. Five results have been found after this study. First, the dynamic schema offered by the NOSQL database is useful when new types of astronomical data need to be archived into the database. With dynamic schema, adding or removing data from a database will not affect the existing database structure. In other words, the database can easily meet the changes in data storage requirements.

Second, the MongoDB supports both complex query and 2D search features that are capable of look up the required astronomical data based on sky coordination from the database.

Third, the NOSQL databases do not support a ‘join’ feature, so it is not possible to lookup different data with a single query command as with SQL. This means that extra programming efforts are required in the application side.

Fourth, the scalability of NOSQL databases is very strong; they use a horizontal scale approach for scaling out the data. Thus, it is possible to boost the capacity of the database system through using multiple commodity machines or low specification servers rather than through adding expensive and powerful hardware on a single machine in order to boost capacity. Moreover, the horizontal scaling approach offers huge benefits when dealing with increased workloads and extremely large dataset situations. When the size of the dataset increases, we can add extra machines to the cluster; NOSQL databases will automatically rebalance the dataset into machines within the cluster. When performing data querying, the NOSQL database has the ability to know which server it needs to access to search for the data. Therefore, the workload of each single server can be reduced significantly. In other words, it is possible to use NOSQL to handle tens of Petabyte of data.

Last, based on the measurement results, the MongoDB offers extremely fast insert and query performance, because it performs all operations on a RAM base. However, it is better to ensure that the working dataset can fit in the RAM in order to avoid cache misses and achieve the best possible performance.

This research provides new ideas for implementing transient notification architecture and managing astronomical data. We have developed a fundamental infrastructure for these two concepts, but there is much more works that can be added on top of this infrastructure in the future. The final outcomes of this research have been very successful and provide exciting directions for resolving similar computational challenges in the astronomical transient field. These ideas can possibly be adapted to other domains for solving similar problems.

References

[1] S. G. Djorgovski, A. A. Mahabal, C. Donalek, M. J. Graham, A. J. Drake, B. Moghaddam, M. Turmon, "Flashes in a Star Stream: Automated Classification of Astronomical Transient Events," *IEEE eScience 2012 conference*, Oct. 2012.

[2] N. Bone, "Observing Meteors, Comets, Supernovae and Other Transient Phenomena," *Astronomy & Space Science*, ISBN 978-1-4471-0579-4, pp. 194-196.

[3] A. Rest, B. Sinnott, D. L. Welch, J. L. Prieto, F. Bianco, "The Echoes of a Supernova," http://astrobites.org/2014/04/02/the-echoes-of-a-supernova/, Apr. 2014.

[4] S. Liebes, "Gravitational Lenses," *Physical Review*, vol.133, pp. 835-844, Feb. 1963.

[5] J. Wambsganss, "Gravitational Microlensing," *ArXiv Astrophysics e-prints*, *astro-ph/0604278*, Apr. 2006.

[6] A. Einstein, "Lens-Like Action of a Star by the Deviation of Light in the Gravitational Field," *Science 4*, vol.84, pp.506-507, Dec. 1936.

[7] R. D. Williams, "Event Handling with SkyAlert," http://www.cacr.caltech.edu/hotwired2/book/chapters/EventHandlingWithSkyalert.pdf, [Accessed: 22-06-2015].

[8] C. H. Ling, "Simulation and Modelling of Gravitational Microlensing Events Using Graphical Processing Units," *PhD Thesis, Massey University*, Jul. 2013.

[9] I.A.Bond, "The First Extrasolar Planet Detected via Gravitational Microlensing," *New Astronomy Reviews*, vol. 56, pp. 25-32, Jan. 2012.

[10] G. Christie, "Detecting Exoplanets by Gravitational Microlensing using a Small Telescope," *ArXiv Astrophysics e-prints, astro-ph/06095998*, Sep. 2006.

[11] N. J. Rattenbury, I. A. Bond, J. Skuljan, P. C. M. Yock, "Planetary Microlensing at High Magnification," *Monthly Notices of the Royal Astronomical Society*, vol. 335, pp. 159-161, May. 2002.

[12] MOA, "Microlensing Alerts," http://www.phys.canterbury.ac.nz/moa/microlensing_alerts.html, [Accessed: 4-29-2014].

[13] B. Paczynki, "Gravitational Microlensing by the Galactic Halo," *The Astrophysical Journal, Part 1*, vol. 304, pp. 1-5, May. 1986.

[14] I. A. Bond, A. Udalski, M. Jaroszyński, N. J. Rattenbury, B. Paczyński, I. Soszyński, L. Wyrzykowski, M. K. Szymański, M. Kubiak, O. Szewczyk, K. Żebruń, G. Pietrzyński, F. Abe, D. P. Bennett, S. Eguchi, Y. Furuta, J. B. Hearnshaw, K. Kamiya, P. M. Kilmartin, Y. Kurata, K. Masuda, Y. Matsubara, Y. Muraki, S. Noda, K. Okajima, T. Sako, T. Sekiguchi, D. J. Sullivan, T. Sumi, P. J. Tristram, T. Yanagisawa, P. C. M. Yock, and The MOA and OGLE Collaborations, "OGLE 2003-BLG-235/MOA 2003-BLG-53: A Planetary Microlensing Event," *The Astrophysical Journal Letters*, J.606:L155-L158, May. 2004.

[15] R. D. Williams, S.G. Djorgovski, A. J. Drake, M. J. Graham, A. Mahabal, “Skyalert: Real-time Astronomy for You and Your Robots,” *Astronomical Data Analysis Software and Systems XVIII ASP Conference Series*, vol. 411, pp.116, Sep. 2009.

[16] C. Han, A. Udalski, J.-Y. Choi, J. C. Yee, A. Gould, G. Christie, T.-G. Tan, M. K. Szymański, M. Kubiak, I. Soszyński, G. Pietrzyński, R. Poleski, K. Ulaczyk, P. Pietrukowicz, S. Kozłowski, J. Skowron, Ł. Wyrzykowski, L. A. Almeida, V. Batista, D. L. Depoy, Subo Dong, J. Drummond, B.S. Gaudi, K.-H. Hwang, F. Jablonski, Y.-K. Jung, C.-U. Lee, J.-R. Koo, J. McCormick, L. A. G. Monard, T. Natusch, H. Ngan, H. Park, R. W. Pogge, Ian Porritt, I.-G. Shin, "The Second Multiple-Planet System Discovered by Microlensing: OGLE-2012-BLG-0026Lb, c---A Pair of Jovian Planets beyond the Snow Line," *The Astrophysical JournalLetter*, vol. 762, pp. L28, Jan. 2013.

[17] D. P. Bennett, T. Sumi, I. A. Bond, K. Kamiya, F. Abe, C. S. Botzler, A. Fukui, K. Furusawa, Y. Itow, A. V. Korpela, P. M. Kilmartin, C. H. Ling, K. Masuda, Y. Matsubara, N. Miyake, Y. Muraki, K. Ohnishi, N. J. Rattenbury, To. Saito, D. J. Sullivan, D. Suzuki, W. L. Sweatman, P. J. Tristram, K. Wada, P. C. M. Yock (The MOA Collaboration),"Planetary and Other Short Binary Microlensing Events from the MOA Short Event Analysis", *The Astrophysical Journal*, vol. 757, pp.119, Oct. 2012.

[18] E. Bachelet, I. G. Shin, C. Han, P. Fouqué, A. Gould, J. W. Menzies, J.-P. Beaulieu, D. P. Bennett, I. A. Bond, Subo Dong, D. Heyrovský, J. B. Marquette, J. Marshall, J. Skowron, R. A. Street, T. Sumi, A. Udalski, L. Abe, K. Agabi, M. D. Albrow, W. Allen, E. Bertin, M. Bos, D. M. Bramich, J. Chavez, G. W. Christie, A. A. Cole, N. Crouzet, S. Dieters, M. Dominik, J. Drummond, J. Greenhill, T. Guillot, C. B. Henderson, F. V. Hessman, K. Horne, M. Hundertmark, J. A. Johnson, U. G. Jørgensen, R. Kandori, C. Liebig, D. Mékarnia, J. McCormick, D. Moorhouse, T. Nagayama, D. Nataf, T. Natusch, S. Nishiyama, J.-P. Rivet, K. C. Sahu, Y. Shvartzvald, G. Thornley, A. R. Tomczak, Y. Tsapras, J. C. Yee, V. Batista, C. S. Bennett, S. Brillant, J. A. R. Caldwell, A. Cassan, E. Corrales, "MOA-2010-BLG-477Lb: constraining the mass of a microlensing planet from microlensing parallax, orbital motion and detection of blended light," *The Astrophysical Journal*, vol. 754, pp. 73, Jul. 2012.

[19] Y. Muraki, C. Han, D.P. Bennett, D. Suzuki, L.A.G. Monard, R. Street, U.G. Jorgensen, P. Kundurthy, J. Skowron, A.C. Becker, M.D. Albrow, P. Fouque, D. Heyrovsky, R.K. Barry, J.-P. Beaulieu, D.D. Wellnitz, I.A. Bond, T. Sumi, S. Dong, B.S. Gaudi, D.M. Bramich, M. Dominik, F. Abe, C.S. Botzler, M. Freeman, A. Fukui, K. Furusawa, F. Hayashi, J.B. Hearnshaw, S. Hosaka, Y. Itow, K. Kamiya, A.V. Korpela, P.M. Kilmartin, W. Lin, C.H. Ling, S. Makita, K. Masuda, Y. Matsubara, N. Miyake, K. Nishimoto, K. Ohnishi, Y.C. Perrott, N.J. Rattenbury, To. Saito, L. Skuljan, D.J. Sullivan, W.L. Sweatman, P.J. Tristram, K. Wada, P.C.M. Yock, G.W. Christie, D.L. DePoy, E. Gorbikov, A. Gould, S. Kaspi, C.-U. Lee, F. Mallia, D. Maoz, J. McCormick, D. Moorhouse, T. Natusch, B.-G. Park, R.W. Pogge, D. Polishook, "Discovery and Mass Measurements of a Cold, 10 Earth Mass Planet and Its Host Star," *The Astrophysical Journal*, vol. 741, pp. 22, Nov. 2011.

[20] V. Batista, A. Gould, S. Dieters, Subo Dong, I.A. Bond, J.P. Beaulieu, D. Maoz, B. Monard, G.W. Christie, J. McCormick, M.D. Albrow, K. Horne, Y. Tsapras, M.J. Burgdorf, S. Calchi Novati, J. Skottfelt, J. Caldwell, S. Kozlowski, D. Kubas, B.S. Gaudi, C. Han, D.P. Bennett, J. An, the MOA Collaboration, the PLANET Collaboration, the MicroFUN Collaboration, the MiNDSTEp Consortium, the RoboNet Collaboration, "MOA-2009-BLG-387Lb: A massive planet orbiting an M dwarf," *Astronomy and Astrophysics*, vol. 529, pp. A102, May. 2011.

[21] N. Miyake, T. Sumi, Subo Dong, R. Street, L. Mancini, A. Gould, D. P. Bennett, Y. Tsapras, J. C. Yee, M. D. Albrow, I. A. Bond, P. Fouque, P. Browne, C. Han, C. Snodgrass, F. Finet, K. Furusawa, K. Harpsoe, W. Allen, M. Hundertmark, M. Freeman, D. Suzuki, F. Abe, C. S. Botzler, D. Douchin, A. Fukui, F. Hayashi, J. B. Hearnshaw, S. Hosaka, Y. Itow, K. Kamiya, P. M. Kilmartin, A. Korpela, W. Lin, C. H. Ling, S. Makita, K. Masuda, Y. Matsubara, Y. Muraki, T. Nagayama, K. Nishimoto, K. Ohnishi, Y. C. Perrott, N. Rattenbury, To. Saito, L. Skuljan, D. J. Sullivan, W. L. Sweatman, P. J. Tristram, K. Wada, P. C. M. Yock, The MOA Collaboration, G. Bolt, M. Bos, G.W. Christie, D.L. DePoy, J. Drummond, A. Gal-Yam, B.S. Gaudi, E. Gorbikov, D. Higgins, K.-H. Hwang J. Janczak, S. Kaspi, C.-U. Lee, J.-R. Koo, "A sub-Saturn Mass Planet, MOA-2009-BLG-319Lb," *The Astrophysical Journal*, vol. 728, pp.120, Feb. 2011.

[22] J. Janczak, A. Fukui, S. Dong, B. Monard, S. Kozlowski, A. Gould, J.P. Beaulieu, D. Kubas, J.B. Marquette, T. Sumi, I. A. Bond, D. P. Bennett, the MOA collaboration, the MicroFUN collaboration, the MiNDSTEp collaboration, the PLANET collaboration,"Sub-Saturn Planet MOA-2008-BLG-310Lb: Likely To Be In The Galactic Bulge," *The Astrophysical Journal*, vol. 711, pp. 731-743, Mar. 2010.

[23] T. Sumi, D.P. Bennett, I.A. Bond, A. Udalski, V. Batista, M. Dominik, P. Fouqué, D. Kubas, A. Gould, B. Macintosh, K. Cook, S. Dong, L. Skuljan, A. Cassan, The MOA Collaboration: F. Abe, C.S. Botzler, A. Fukui, K. Furusawa, J.B. Hearnshaw, Y. Itow, K. Kamiya, P.M. Kilmartin, A. Korpela, W. Lin, C.H. Ling, K. Masuda, Y. Matsubara, N. Miyake, Y. Muraki, M. Nagaya, T. Nagayama, K. Ohnishi, T. Okumura, Y.C. Perrott, N. Rattenbury, To. Saito, T. Sako, D.J. Sullivan, W.L. Sweatman, P., P.C.M. Yock, The PLANET Collaboration: J.P. Beaulieu, A. Cole, Ch. Coutures, M.F. Duran, J. Greenhill, F. Jablonski, U. Marboeuf, E. Martioli, E. Pedretti, O. Pejcha, P. Rojo, M.D. Albrow, S. Brillant, M. Bode, D.M. Bramich, M.J. Burgdorf, J.A.R. Caldwell, H. Calitz, E. Corrales, S. Dieters, D. Dominis Prester, "A Cold Neptune-Mass Planet OGLE-2007-BLG-368Lb: Cold Neptunes Are Common," *The Astrophysical Journal*, vol. 710, pp. 1641-1653, Feb. 2010.

[24] S. Dong, I.A. Bond, A. Gould, S. Koz lowski, N. Miyake, B.S. Gaudi,D.P. Bennett, F. Abe, A.C. Gilmore, A. Fukui, K. Furusawa, J.B. Hearnshaw, Y. Itow, K. Kamiya,P.M. Kilmartin, A. Korpela, W. Lin, C.H. Ling, K. Masuda, Y. Matsubara,Y. Muraki, M. Nagaya, K. Ohnishi, T. Okumura, Y.C. Perrott, N. Rattenbury,To. Saito, T. Sako, S. Sato, L. Skuljan, D.J. Sullivan, T. Sumi, W. Sweatman,P.J. Tristram, P.C.M. Yock,(The MOA Collaboration)G. Bolt, G.W. Christie, D.L. DePoy, C. Han, J. Janczak, C.-U. Lee, F. Mallia, J. McCormick, B. Monard, A. Maury, T. Natusch, B.-G. Park, R.W. Pogge,R. Santallo, K.Z. Stanek(The µ FUN Collaboration),A. Udalski, M. Kubiak, M.K. Szyma´nski, G. Pietrzy´nski,, I. Soszy´nski,O. Szewczyk,, L. Wyrzykowski,, K. Ulaczyk, (The OGLE Collaboration),"Microlensing Event MOA-2007-BLG-400: Exhuming the Buried Signature of a Cool, Jovian-Mass Planet," *The Astrophysics Journal*, vol. 698, pp. 1826-1837, Jun. 2009.

[25] D. Kubas, J. P. Beaulieu, D.P. Bennett, A. Cassan, A. Cole, J. Lunine, J.B. Marquette, S. Dong, A. Gould, T. Sumi, V. Batista, P. Fouque, S. Brillant, S. Dieters, C. Coutures, J. Greenhill, I. Bond, T. Nagayama, A.Udalski, E. Pompei, D.E.A. Nuernberger, J.B. Le Bouquin, "A Frozen Super-Earth Orbiting a Star at the Bottom of the Main Sequence," *Astronomy and Astrophysics*, vol. 540, pp. A78, Apr.2012.

[26] D.P. Bennett, I.A. Bond, A. Udalski, T. Sumi, F. Abe, A. Fukui, K. Furusawa, J.B. Hearnshaw, S. Holderness, Y. Itow, K. Kamiya, A.V. Korpela, P.M. Kilmartin, W. Lin, C.H. Ling, K. Masuda, Y. Matsubara, N. Miyake, Y. Muraki, M. Nagaya, T. Okumura, K. Ohnishi, Y.C. Perrott, N.J. Rattenbury, T. Sako, To. Saito, S. Sato, L. Skuljan, D.J. Sullivan, W.L. Sweatman, P.J. Tristram, P.C.M. Yock, M. Kubiak, M.K. Szymanski, G. Pietrzynski, I. Soszynski, O. Szewczyk, L. Wyrzykowski, K. Ulaczyk, V. Batista, J.P. Beaulieu, S. Brillant, A. Cassan, P. Fouque, P. Kervella, D. Kubas, J.B. Marquette, "A Low-Mass Planet with a Possible Sub-Stellar-Mass Host in Microlensing Event MOA-2007-BLG-192," *The Astrophysical Journal*, vol. 684, no.1, pp. 663-683, Sep.2008.

[27] D.P. Bennett, S.H. Rhie, S. Nikolaev, B.S. Gaudi, A. Udalski, A. Gould, G.W. Christie, D. Maoz, S. Dong, J. McCormick, M.K. Szymanski, P.J. Tristram, B. Macintosh, K.H. Cook, M. Kubiak, G. Pietrzynski, I. Soszynski, O. Szewczyk, K. Ulaczyk, L. Wyrzykowski, D.L. DePoy, C. Han, S. Kaspi, C.-U. Lee, F. Mallia, T. Natusch, B.-G. Park, R.W. Pogge, D. Polishook, F. Abe, I.A. Bond, C.S. Botzler, A. Fukui, J.B. Hearnshaw, Y. Itow, K. Kamiya, A.V. Korpela, P.M. Kilmartin, W. Lin, K. Masuda, Y. Matsubara, M. Motomura, Y. Muraki, S. Nakamura, T. Okumura, K. Ohnishi, Y.C. Perrott, N.J. Rattenbury, T. Sako, To. Saito, S. Sato, L. Skuljan, D.J. Sullivan, T. Sumi, W.L. Sweatman, P.C.M. Yock, M. Albrow, A. Allan, J.-P. Beaulieu, D.M. Bramich, M.J. Burgdorf, C. Coutures, M. Dominik, S. Dieters, P. Fouque, "Masses and Orbital Constraints for the OGLE-2006-BLG-109Lb,c Jupiter/Saturn Analog Planetary System," vol.713, pp. 837-855, Apr. 2010.

[28] B.S.Gaudi, D.P.Bennett, A.Udalski, A.Gould, G.W.Christie, D.Maoz, S.Dong, J.McCormick, M.K.Szymanski, P.J.Tristram, S.Nikolaev, B.Paczynski, M.Kubiak, G.Pietrzynski, I.Soszynski, O.Szewczyk, K.Ulaczyk, L.Wyrzykowski, D.L.DePoy, C.Han, S.Kaspi, C.-U.Lee, F.Mallia, T.Natusch, R.W.Pogge, B.-G.Park, F.Abe, I.A.Bond, C.S.Botzler, A.Fukui, J.B.Hearnshaw, Y.Itow, K.Kamiya, A.V.Korpela, P.M.Kilmartin, W.Lin, K.Masuda, Y.Matsubara, M.Motomura, Y.Muraki, S.Nakamura, T.Okumura, K.Ohnishi, N.J.Rattenbury, T.Sako, To.Saito, S.Sato, L.Skuljan, D.J.Sullivan, T.Sumi, W.L.Sweatman, P.C.M.Yock, M.D.Albrow, A.Allan, J.-P.Beaulieu, M.J.Burgdorf, K.H.Cook, C.Coutures, M.Dominik, S.Dieters, P.Fouque, J.Greenhill, K.Horne, I.Steele, Y.Tsapras, B.Chaboyer, A.Crocker, S.Frank, B.Macintosh (OGLE, MicroFUN, MOA, PLANET/RoboNET), "Discovery of a Jupiter/Saturn Analog with Gravitational Microlensing," *Science*, vol. 319, pp. 927, Mar. 2008.

[29] S. Dong, A. Gould, A. Udalski, J. Anderson, G.W. Christie, B.S. Gaudi, M. Jaroszynski, M. Kubiak, M.K. Szymanski, G. Pietrzynski, I. Soszynski, O. Szewczyk, K. Ulaczyk, L. Wyrzykowski, D.L. DePoy, D.B. Fox, A. Gal-Yam, C. Han, S. Lepine, J. McCormick, E. Ofek, B.G. Park, R.W. Pogge, F. Abe, D.P. Bennett, I.A. Bond, T.R. Britton, A.C. Gilmore, J.B. Hearnshaw, Y. Itow, K. Kamiya, P.M. Kilmartin, A. Korpela, K. Masuda, Y. Matsubara, M. Motomura, Y. Muraki, S. Nakamura, K. Ohnishi, C. Okada, N. Rattenbury, To. Saito, T. Sako, M. Sasaki, D. Sullivan, T. Sumi, P.J. Tristram, T. Yanagisawa, P.C.M. Yock, T. Yoshoika, M.D. Albrow, J.P. Beaulieu, S. Brillant, H. Calitz, A. Cassan, K. H. Cook, Ch. Coutures, S. Dieters, D. Dominis Prester, J. Donatowicz, P. Fouque, J. Greenhill, K. Hill, "OGLE-2005-BLG-071Lb, the Most Massive M-Dwarf Planetary Companion?," *The Astrophysical Journal*, vol. 695, pp. 970-987, Apr. 2009.

[30] A. Gould, A. Udalski, D. An, D.P. Bennett, A.-Y. Zhou, S. Dong, N.J. Rattenbury, B.S. Gaudi, P.C.M. Yock, I.A. Bond, G.W. Christie, K. Horne, J. Anderson, K.Z. Stanek, D.L. DePoy, C. Han, J. McCormick, B.-G. Park, R.W. Pogge, S.D. Poindexter, I. Soszynski, M.K. Szymanski, M. Kubiak, G. Pietrzynski, O. Szewczyk, L. Wyrzykowski, K. Ulaczyk, B. Paczynski, D.M. Bramich, C. Snodgrass, I.A. Steele, M.J. Burgdorf, M.F. Bode, C.S. Botzler, S. Mao, S.C. Swaving (The MicroFUN, OGLE, and PLANET/RoboNet collaborations), "Microlens OGLE-2005-BLG-169 Implies Cool Neptune-Like Planets are Common," *The Astrophysical Journal*, vol. 644, pp. L37-L40, Jun. 2006.

[31] J.-P. Beaulieu, D.P. Bennett, P. Fouque, A. Williams, M. Dominik, U.G. Jorgensen, D. Kubas, A. Cassan, C. Coutures, J. Greenhill, K. Hill, J. Menzies, P.D. Sackett, M. Albrow, S. Brillant, J.A.R. Caldwell, J.J. Calitz, K.H. Cook, E. Corrales, M. Desort, S. Dieters, D. Dominis, J. Donatowicz, M. Hoffman, S. Kane, J.-B. Marquette, R. Martin, P. Meintjes, K. Pollard, K. Sahu, C. Vinter, J. Wambsganss, K. Woller, K. Horne, I. Steele, D. Bramich, M. Burgdorf, C. Snodgrass, M. Bode (PLANET) A. Udalski, M. Szymanski, M. Kubiak, T. Wieckowski, G. Pietrzynski, I. Soszynski, O. Szewczyk, L. Wyrzykowski, B. Paczynski (OGLE), the MOA Collaboration, "Discovery of a Cool Planet of 5.5 Earth Masses Through Gravitational Microlensing," *Nature International Weekly Journal of Science*, vol.439, pp. 437-440, Jan. 2006.

[32] A. Udalski, M. Jaroszynski, B. Paczynski, M. Kubiak, M.K. Szymanski, I. Soszynski, G. Pietrzynski, K. Ulaczyk, O. Szewczyk, L. Wyrzykowski (The OGLE Collaboration), G.W. Christie, D.L. DePoy, S. Dong, A. Gal-Yam, B.S. Gaudi, A. Gould, C. Han, S. Lepine, J. McCormick, B.-G. Park, R.W. Pogge (The microFUN Collaboration), D.P. Bennett, I.A. Bond, Y. Muraki, P.J. Tristram, P.C.M.Yock (From the MOA Collaboration), J.P. Beaulieu, D.M. Bramich, S.W. Dieters, J. Greenhill, K. Hill, K. Horne, D. Kubas (From the PLANET/RoboNet Collaboration), "A Jovian-mass Planet in Microlensing Event OGLE-2005-BLG-071," *The Astrophysical Journal*, vol.628, pp. L109-L112, Aug. 2005.

[33] D. P. Bennett, J. Anderson, I. A. Bond, A. Udalski, A. Gould, "Identification of the OGLE-2003-BLG-235/MOA-2003-BLG-53 Planetary Host Star", *The Astrophysical Journal*, vol.647, pp. L171-L174, Aug. 2006.

[34] R. Nishi, K. Ioka, Y. Kan-ya, "Dark Matter Search with Gravitational Microlensing Events," *Progress of Theoretical Physic Supplement*, vol.133, pp. 211-231, 1999.

[35] N. F. Bate, D. J. E. Floyd, R. L. Webster, J. S. B. Wyithe, "A microlensing measurement of dark matter fractions in three lensing galaxies," *The Astrophysics Journal*, vol.731, pp. 1, 2011.

[36] K. Griest, C. Alcock, R. A. Allsman, T. S. Axelrod, D. P. Bennett, K. H. Cook, K. C. Freeman, J. Guern, M. Lehner, S. L. Marshall, S. Perlmutter, B. A. Peterson, M. R. Pratt, P J Quinn, A. W. Rodgers, C. W. Stubbs, W. Sutherland, D. Welch, "MACHO Collboration Search for Baryonic Dark Matter via Gravitational Microlensing," *Arxiv Astrophysics e-prints, astro-ph/9506016*, pp. 1-6, Jun. 1995.

[37] R. Massey, T. Kitching, J. Richard, "The Dark Matter of Gravitational Lensing," *The Astrophysics Journal*, vol.73, pp. 2-8, Jul. 2010.

[38] B.Paczynski, "Gravitataional Microlensing: Black Holes, Planets: OGLE, VLTI, HST and Space Probes," *ArXiv Astrophysics e-prints, astro-ph/0306564*, Jun. 2003.

[39] I. O'Neill, "Hunting Black Holes through a Gravitational Lens," http://news.discovery.com/space/hunting-black-holes-with-gravitational-lenses-120210.htm, Feb. 2012.

[40] E. Agol, M. Kamionkowski, Léon V. E. Koopmans, R. D. Blandford, "Finding Black Holes with Microlensing," *The Astrophysical Journal*, Vol. 576, pp. L131-L135, Sep. 2002.

[41] MOA, "Microlensing Observations in Astrophysics (MOA)," http://www.sciencelearn.org.nz/Contexts/Space-Revealed/NZ-Research/Microlensing-Observations-in-Astrophysics-MOA, [Accessed: Apr. 2014].

[42] T. Sako, T. Sekiguchi, M. Sasaki, K. Okajima, F. Abe, I. A. Bond, J. B. Hearnshaw, Y. Itow, K. Kamiya, P. M. Kilmartin, K. Masuda, Y. Matsubara,Y. Muraki, N. J. Rattenbury, D. J. Sullivan, T. Sumi, P. Tristram, T. Yanagisawa, P. C. M. Yock, "MOA-cam3: A Wide-Feld Mosaic CCD Camera for a Gravitational Microlensing Survey in New Zealand," *Experimental Astronomy*, vol. 22, pp. 51-66, Dec. 2007.

[43] J. B. Hearnshaw, F. Abe, I.A. Bond, A.C. Gilmore, Y. Itow, K. Kamiya,K. Masuda, Y. Matsubara, Y. Muraki, C. Okada, N.J. Rattenbury, T. Sako,M. Sasaki, D.J. Sullivan, P.C.M. Yock, "The MOA 1.8-metre alt-az Wide-Field Survey Telescope and the MOA Project," *ArXiv Astrophysics e-prints, astro-ph/0509420,* pp.1-2, Feb. 2008.

[44] J. Wambsganss, “Gravitational Lensing in Astronomy,” *Living Rev. Relativity*, vol.1, pp. 8-9, Nov. 1998.

[45] C.Alcock, R.Allsman, T.Axelrod, D.Bennett, S.Chan, K.Cook, K.Freeman, K.Griest, S. Marshall, S.Perlmutter, B.Peterson, M.Pratt, P.Quinn, A.Rodgers, C.Stubbs, W.Sutherland, “Probable Gravitational Microlensing towards the Galatic Bulge,” *The Astrophysical Journal*, Part 1, vol.445, pp. 133-139, Mar. 1995.

[46] MACHO, “MACHO Website,” http://wwwmacho.anu.edu.au/, [Accessed: 07-05-2014].

[47] EROS, “EROS Website,” http://eros.in2p3.fr/, [Accessed: 07-05-2014].

[48] C. Alcock, R. A. Allsman, D. Alves, T. S. Axelrod, A. C. Becker, D. P. Bennett, K. H. Cook, K. C. Freeman, K. Griest, J. Guern, M. J. Lehner, S. L. Marshall, B. A. Peterson, M. R. Pratt, P. J. Quinn, A. W. Rodgers, C. W. Stubbs, W. Sutherland, D. L. Welch (The MACHO Collaboration), “The MACHO Project Large Magellanic Cloud Microlensing Results from the First Two Years and the Nature of the Galactic Dark Halo,” *The Astrophysical Journal*, Vol.486, pp. 697-726, Sep. 1997.

[49] C. Alcock, R. A. Allsman, T. S. Axelrod, D. P. Bennett, K. H. Cook, K. C.Freeman, K. Griest, J. A. Guern, M. J. Lehner, S. L. Marshall, H. -S. Park, S. Perlmutter, B. A. Peterson, M. R. Pratt, P. J. Quinn, A. W. Rodgers, C. W. Stubbs, W. Sutherland (The MACHO Collaboration), “The MACHO Project First Year LMC Results: The Microlensing Rate and the Nature of the Galactic Dark Halo,” *The Astrophysical Journal*, vol.461, pp. 84, Apr. 1996.

[50] O. Perdereau, “First Results from the EROS-II Microlensing Experiment,” *The Third Stromlo Symposium: The Galactic Halo ASP Conference Series*, vol. 165, pp. 370, 1999.

[51] C. Alcock, C. W. Akerlof, R. A. Allsman, T. S. Axelrod, D. P. Bennett, S. Chan, K. H. Cook, K. C. Freeman, K. Griest, S. L. Marshall, H-S. Park, S. Perlmutter, B. A. Peterson, M. R. Pratt, P. J. Quinn, A. W. Rodgers, C. W. Stubbs, W. Sutherland, (The MACHO Collaboration), “Possible Gravitational Microlensing of a Star in the Large Magellanic Cloud,” *Nature*, vol.365, pp. 621 – 623, Oct. 1993.

[52] E. Aubourg, P. Bareyre, S. Bréhin, M. Gros, M. Lachièze-Rey, B. Laurent, E. Lesquoy, C. Magneville, A. Milsztajn, L. Moscoso, F. Queinnec, J. Rich, M. Spiro, L. Vigroux, S. Zylberajch, R. Ansari, F. Cavalier, M. Moniez, J.-P. Beaulieu, R. Ferlet, Ph. Grison, A. Vidal-Madjar, J. Guibert, O. Moreau, F. Tajahmady, E. Maurice, L. Prévôt, C. Gry, “Evidence for Gravitational Microlensing by Dark Objects in the Galactic Halo,” *Nature*, vol.365, pp. 623-625, Oct. 1993.

[53] P. Tisserand, L. Le Guillou, C. Afonso, J.N. Albert, J. Andersen, R. Ansari, E. Aubourg, P. Bareyre, J.P. Beaulieu, X. Charlot, C. Coutures, R. Ferlet, P. Fouqué, J.F. Glicenstein, B. Goldman, A. Gould, D. Graff, M. Gros, J. Haissinski, C. Hamadache, J. de Kat, T. Lasserre, E. Lesquoy, C. Loup, C. Magneville, J.B. Marquette, E. Maurice, A. Maury, A. Milsztajn, M. Moniez, N. Palanque-Delabrouille, O. Perdereau, Y.R. Rahal, J. Rich, M. Spiro, A. Vidal-Madjar, L. Vigroux, S. Zylberajch, "Limits on the Macho Content of the Galactic Halo from the EROS-2 Survey of the Magellanic Clouds," *Astronomy and Astrophysics*, vol. 469, pp. 387-404, Jul. 2007.

[54] A. Udalski, M. Szymanski, J. Kaluzny, M. Kubiak, M. Mateo, W. Krzeminski, "The Optical Gravitational Lensing Experiment: The Discovery of Three Further Microlensing Events in the Direction of the Galactic Bulge," *The Astrophysical Journal*, vol. 426, pp. L69-L72, May. 1994.

[55] OGLE, "OGLE," http://ogle.astrouw.edu.pl/, [Accessed: 25-05-2014].

[56] A. Udalski, M. Szymanski, J. Kaluzny, M. Kubiak, W. Krzeminski, M. Mateo, "The Optical Gravitational Lensing Experiment. Discovery of the First Candidate Microlensing Event in the Direction of the Galactic Bulge," *Acta Astronomica*, vol. 43, pp. 289–294, Jun. 1996.

[57] A. Udalski, M. Szymanski, J. Kaluzny, M. Kubiak, M. Mateo, W. Krzeminski, B. Paczynski, "The Optical Gravitational Lensing Experiment. The Early Warning System," *Acta Astronomica*, vol. 44, pp. 227–234, Aug. 1994.

[58] B. Paczynski, K. Z. Stanek, A. Udaliski, M. Szymanski, J. Kaluzny, M. Kubiak, M. Mateo, W. Krzeminski, G. W. Preston, "Results from the Optical Gravitational Lensing Experiment (OGLE)," *169th Symposium of the International Astronomical Union*, pp. 93, Nov. 1996.

[59] A. Udalski, M. Kubiak, M. Szymanski, "Optical Gravitational Lensing Experiment. OGLE-2 -- the Second Phase of the OGLE Project," *Acta Astronomica*, vol. 47, pp. 319–344, Oct. 1997.

[60] K. Zebrun, A. Udalski, M. Szymanski, M. Kubiak, G. Pietrzynski, I. Soszynski, P. Wozniak, "The Optical Gravitational Lensing Experiment OGLE-II Results," *Gravitation and Relativistic Field Theories*, vol.3, pp. 2147-2148, Jul. 2000.

[61] M. Szymański, A. Udalski, M. Kubiak, G. Pietrzyński, I. Soszyński, K. Zebruń, P. Wozniak, "OGLE Survey - Microlensing in the Milky Way," *Dynamics of Star Clusters and the Milky Way ASP Conference Series*, vol.228, 2001.

[62] G. Pietrzynski, A. Udalski, "OGLE II Observations of Star Clusters in the Magellanic Clouds," *Extragalactic Star Clusters IAU Symposium Series*, vol. 207, pp.184, 2002.

[63] A. Udalski, "The Optical Gravitational Lensing Experiment. Real Time Data Analysis Systems in the OGLE-III Survey," *Acta Astronomica*, vol. 53, pp. 291-305, Jan. 2004.

[64] OGLE, "OGLE Early Warning System," http://ogle.astrouw.edu.pl/ogle4/ews/ews.html, [Accessed: 20-05-2014].

[65] A. Udalski, "XROM and RCOM: Two New OGLE-III Real Time Data Analysis Systems," *Acta Astronomica*, vol.58, pp. 187-192, Oct. 2008.

[66] A. Udalski, "Transit Campaigns of the OGLE-III Survey," *Transiting Extrasolar Planets Workshop ASP Conference Series*," vol. 366, pp.51 2007.

[67] A.Udalski, "Status of the OGLE-IV Project," http://smc2011.physics.unisa.it/talks/udaski.pdf, [Accessed: 21-05-2014].

[68] OGLE, "OGLE Photometry Database," http://ogledb.astrouw.edu.pl/~ogle/photdb/, [Accessed: 21-05-2014].

[69] MicroFun, "MicroFun Website," http://www.astronomy.ohio-state.edu/~microfun/, [Accessed: 21-05-2014].

[70] MiNDSTEp, "MiNDSTEp Website," http://www.mindstep-science.org/, [Accessed: 11-06-2015].

[71] Korea Astronomy and Space Science Institute, "Call for Science Proposals for Korea Microlensing Telescope Network (KMTNet)," May. 2012.

[72] V. Krabbendam, "Project and Technical Overview," *LSST All Hands Meeting*, Aug. 2012.

[73] Z. Ivezic, J. A. Tyson, E. Acosta, R. Allsman, S. F. Anderson, J. Andrew, R. Angel, T. Axelrod, J. D. Barr, A. C. Becker, J. Becla, C. Beldica, R. D. Blandford, J. S. Bloom, K. Borne, W. N. Brandt, M. E. Brown, J. S. Bullock, D. L. Burke, S. Chandrasekharan, S. Chesley, C. F. Claver, A. Connolly, K. H. Cook, A. Cooray, K. R. Covey, C. Cribbs, R. Cutri, G. Daues, F. Delgado, H. Ferguson, E. Gawiser, J. C. Geary, P. Gee, M. Geha, R. R. Gibson, D. K. Gilmore, W. J. Gressler, C. Hogan, M. E. Huffer, S. H. Jacoby, B. Jain, J. G. Jernigan, R. L. Jones, M. Juric, S. M. Kahn, J. S. Kalirai, J. P. Kantor, R. Kessler, D. Kirkby, L. Knox, V. L. Krabbendam, S. Krughoff, S. Kulkarni, R. Lambert, D. Levine, M. Liang, K-T. Lim, R. H. Lupton, P. Marshall, S. Marshall, M. May, M. Miller, D. J. Mills, D. G. Monet, D. R. Neill, M. Nordby, "LSST: from Science Drivers to Reference Design and Anticipated Data Products," *ArXiv Astrophysics e-prints, 0805.2366*, May. 2008.

[74] LSST, "LSST Reference Design," http://www.lsst.org/files/docs/LSST-RefDesign.pdf, [Accessed: 23-05-2014].

[75] V. Krabbendam, W. J. Gressler, J. R. Andrew, J. D. Barr, J. DeVries, E. Hileman, M. Liang, D. R. Neill, J. Sebag, O. Wiecha, LSST Collaboration, "LSST Telescope and Optics Status," *American Astronomical Society*, vol. 43, 2011.

[76] M. Stephens, "Petabyte-Chomping Big Sky Telescope Sucks down Baby Code," http://www.theregister.co.uk/Print/2010/11/26/lsst_big_data_and_agile/, Nov. 2010, [Accessed: 23-05-2014].

[77] M. Stephens, "Mapping the Universe at 30 Terabytes a Night," http://www.theregister.co.uk/Print/2008/10/03/lsst_jeff_kantor/, Oct. 2008, [Accessed: 23-05-2014].

[78] LSST, "LSST and Technology Innovation," http://www.lsst.org/lsst/science/technology, [Accessed: 23-05- 2014].

[79] CSS, "Catalina Sky Survey Facilities," http://www.lpl.arizona.edu/css/css_facilities.html, [Accessed: 28-05-2014]

[80] A. J. Drake, S. G. Djorgovski, A. Mahabal, E. Beshore, S. Larson, M. J. Graham, R. Williams, E. Christensen, M. Catelan, A. Boattini, A. Gibbs, R. Hill, R. Kowalski, "First Results from the Catalina Real-time Transient Survey," *The Astrophysical Journal*, vol.696, pp. 870-884, May. 2009.

[81] S. G. Djorgovski, A. J. Drake, A. A. Mahabal, M. J. Graham, C. Donalek, R. Williams, E. C. Beshore, S. M. Larson, J. Prieto, M. Catelan, E. Christensen, R. H. McNaught, "The Catalina Real-Time Transient Survey (CRTS)," *Arxiv Astrophysics e-print: 1102.5004*, Feb. 2011.

[82] A. J. Drake, S. G. Djorgovski, A. Mahabal, J. L. Prieto, E. Beshore, M. J. Graham, M. Catalan, S. Larson, E. Christensen, C. Donalek, R. Williams, "The Catalina Real-time Transient Survey," *New Horizons in Time Domain Astronomy, Proceedings of the International Astronomical Union, IAU Symposium*, vol. 285, pp. 306-308, Apr. 2012.

[83] The CRTS Survey, "CTRS Transient Discoveries," http://crts.caltech.edu/index.html, [Accessed: 10-06-2014].

[84] S. G. Djorgovski, A. Drake, A. Mahabal, C. Donalek, R. Williams, M. Graham, E. Beshore, S. Larson, "Discovery, Classification, and Scientific Exploration of Transient Events from the Catalina Real-time Transient Survey," *Bulletin of the Astronmical Society of India*, vol. 39, pp. 387-408, Sep. 2009.

[85] NASA Goddard Space Flight Center, "WFIRST," http://wfirst.gsfc.nasa.gov/, [Accessed: 10-06-2014].

[86] D. Spergel, N. Gehrels, C. Baltay, D. Bennett, J. Breckinridge, M. Donahue, A. Dressler, B. S. Gaudi, T. Greene, O. Guyon, C. Hirata, J. Kalirai, N. J. Kasdin, B. Macintosh, W. Moos, S. Perlmutter, M. Postman, B. Rauscher, J. Rhodes, Y. Wang, D. Weinberg, D. Benford, M. Hudson, W. -S. Jeong, Y. Mellier, W. Traub, T. Yamada, P. Capak, J. Colbert, D. Masters, M. Penny, D. Savransky, D. Stern, N. Zimmerman, R. Barry, L. Bartusek, K. Carpenter, E. Cheng, D. Content, F. Dekens, R. Demers, K. Grady, C. Jackson, G. Kuan, J. Kruk, M. Melton, B. Nemati, B. Parvin, I. Poberezhskiy, C. Peddie, J. Ruffa, J. K. Wallace, A. Whipple, E. Wollack, F. ZhaoWikipedia, "Wide-Field InfrarRed Survey Telescope-Astrophysics Focused Telescope Assets WFIRST-AFTA 2015 Report," *Arxiv Astrophysics e-prints:1503.03757*," Mar. 2015.

[87] P. Hertz, "NASA Townhall AAS 222nd Meeting Indianapolis," http://science.nasa.gov/media/medialibrary/2013/06/04/AAS_TownHall_2013-06_V7A_for_posting.pdf, June. 2013, [Accessed: 13-06-2014].

[88] IVOA, "IVOA Website," http://www.ivoa.net/index.html, [Accessed: 15-06-2014].

[89] IVOA, “What is the IVOA,” http://www.ivoa.net/about/what-is-ivoa.html, [Accessed: 15-06-2014].

[90] C. Arviset, S. Gaudet, IVOA Technical Coordination Group, “IVOA Architecture Version 1.0,” http://www.ivoa.net/documents/Notes/IVOAArchitecture/20101123/IVOAArchitecture-1.0-20101123.pdf, Nov. 2010.

[91] R. Hanisch, P. Quinn, “The International Virtual Observatory,” http://www.ivoa.net/about/TheIVOA.pdf, [Accessed: 15-06-2014].

[92] Wikipedia, “International Virtual Observatory Alliance,” http://en.wikipedia.org/wiki/IVOA, [Accessed: 15-06- 2014].

[93] M. H. Malayeri, N. Moreau, F. Le Petit, “A Novel Semantic Software for Astronomical Concepts,” *International Astronomical Union Information Bulletin*, Jun. 2012.

[94] IVOA, “Introduction to VO Concepts”, http://www.ivoa.net/deployers/intro_to_vo_concepts.html, [Accessed: 16-06-2014].

[95] F. Ochsenbein, R. Williams, C. Davenhall, D. Durand, P. Fernique, D. Giaretta, R. Hanisch, T. McGlynn, A. Szalay, M.Taylor, A. Wicenec, “VOTable Format Definition Version 1.3,” http://wiki.ivoa.net/twiki/bin/view/IVOA/IvoaVOTable, [Accessed: 16-06-2014].

[96] FITS, “A Primer on the FITS Data Format,” http://fits.gsfc.nasa.gov/fits_primer.html, [Accessed: 18-06-2014].

[97] R. Seaman, R. Williams, A. Allan, S. Barthelmy, J. Bloom, J. Brewer, R. Denny, M. Fitzpatrick, M. Graham, N. Gray, F. Hessman, S. Marka, A. Rots, T. Vestrand, P. Wozniak, “IVOA Recommendation: Sky Event Reporting Metadata Version 2.0,” *Arxiv Astrophysics e-print: 1110.0523*, Oct. 2011, [Accessed: 11-07-2014].

[98] R. D. Williams, R. Seaman, “VOEvent: Information Infrastructure for Real-Time Astronomy,” *Astronomical Data Analysis Software and Systems XV ASP Conference Series*, vol.351, pp. 637, Jul. 2006.

[99] R. Seaman, “What is VOEvent?,” http://www.cacr.caltech.edu/hotwired2/book/chapters/WhatIsVOEvent.pdf, [Accessed: 11-07-2014].

[100] VOEventNet, “VOEventNet Website,” http://voeventnet.caltech.edu/, [Accessed: 15-07-2014].

[101] M. J. Graham (Caltech), “VOEventNet,” wiki.ivoa.net/internal/IVOA/VoeventWorkshop2/VOEventII.ppt, [Accessed: 18-07-2014].

[102] T. Budavári, “Probabilistic Cross- identification of Cosmic Events,” *The Astrophysical Journal*, vol. 736, pp. 155-160, Aug. 2011.

[103] A. Mahabal, S. G. Djorgovski, R. Williams, A. Drake, C. Donalek, M. Graham (Caltech), B. Moghaddam, M. Turmon, J. Jewell (JPL), A. Khosla, B. Hensley (Caltech), "Towards Real-time Classification of Astronomical Transients," *Classification and Discovery in Large Astronomical Surveys: Proceedings of the International Conference: Classification and Discovery in Large Astronomical Surveys, AIP Conference Proceedings*, vol. 1082, pp. 287-293, Dec. 2008.

[104] Skyalert, "Skyalert Website," http://www.skyalert.org/, [Accessed: 28-07-2014].

[105] R. D. Williams, S. D. Barthelmy, R. B. Denny, M. J. Graham, J. Swinbank, "Responding to the Event Deluge," Observatory Operations: Strategies, Processes, and Systems IV, vol. 8448, Sep. 2012.

[106] R. Seaman, R. Williams, M. Graham, T. Murphy, "Using the VO to Study the Time Domain," New Horizons in Time-Domain Astronomy, Proceedings of the International Astronomical Union, IAU Symposium, vol. 285, pp. 221-226, Apr. 2012.

[107] GCN, "The Gamma-ray Coordination Network (GCN)," http://gcn.gsfc.nasa.gov/about.html, [Accessed: 05-08-2014].

[108] S. Barthelmy, "VO-GCN/TAN: past, present, and future. Serving the transient communities needs," *Hot Wiring the Transient Universe*, vol. 3, pp. 118-173.

[109] GCN, "GCN-An Invitation to New Sites," http://gcn.gsfc.nasa.gov/invitation.html, [Accessed: 05-08-2014].

[110] High Energy Transient Explorer, "HETE-2 Spacecraft," http://space.mit.edu/HETE/spacecraft.html, [Accessed: 05-08-2014].

[111] Integral, "International Gamma-Ray Astrophysics Laboratory," http://sci.esa.int/integral/, [Accessed: 05-08-2014].

[112] NASA, "Swift Gamma-Ray Burst Mission," http://swift.gsfc.nasa.gov/, [Accessed: 05-08-2014].

[113] ASI – Agenzia Spaziale Italiana, "Astro-Rivelatore Gamma a Immagini Leggero," http://www.asi.it/en/activity/high_energy/agile, [Accessed: 05-08-2014].

[114] NASA, "Fermi Gamma-ray Space Telescope," http://fermi.gsfc.nasa.gov/, [Accessed: 05-08-2014].

[115] NASA, "The GCN Circulars," http://gcn.gsfc.nasa.gov/gcn_circulars.html, [Accessed: 06-08-2014].

[116] NASA, "The GCN Report," http://gcn.gsfc.nasa.gov/reports.html, [Accessed: 06-08-2014].

[117] NOAO, "National Optical Astronomy Observatory," http://www.noao.edu/, [Accessed: 07-08-2014].

[118] S. Barthelmy, "VO-GCN Status," Apr. 2009, http://www.cacr.caltech.edu/hotwired2/program/presentations/HTU2_VOGCN_Barthelmy.pdf, [Accessed: 08-08-2014].

[119] Zen, "Long polling VS Short Polling," http://codertalks.com/long-polling-vs-short-polling/, Sep. 2012, [Accessed: 10-08-2014].

[120] M. Balliauw, "Techniques for Real-time Client-Server Communication on the Web,"http://blog.maartenballiauw.be/post/2011/11/29/Techniques-for-real-time-client-server-communication.aspx, Nov. 2011, [Accessed: 10-08-2014].

[121] J. Lengstorf, P. Leggetter, "Realtime Web Apps with HTML5 WebSocket, PHP, and jQuery," *ISBN13:978-1-4302-4620-6*, Apr. 2013.

[122] E. Bozdag, A. Mesbah, A. van Deursen, "A Comparison of Push and Pull Techniques for AJAX, *Delft University of Technology Software Engineering Research Group Technical Report Series, ISSN: 1872-5392*, http://arxiv.org/pdf/0706.3984.pdf.

[123] E. Stratmann, J. Ousterhout, S. Madan, "Integrating Long Polling with an MVC Web Framework," *WebApps'11 Proceedings of the 2nd USENIX conference on Web application development*, pp. 10, Jun. 2011.

[124] M. Franklin, S. Zdonik, "Data in Your Face: Push Technology in Perspective," *SIGMOD '98 Proceedings of the 1998 ACM SIGMOD international conference on Management of data, ISBN: 0-89791-995-5*, pp. 516-519, Jun. 1998.

[125] P. Deolase, A. Katkar, A. Panchbudhe, K. Ramamritham, P. Shenoy, "Adaptive Push-Pull: Disseminating Dynamic Web Data," *IEEE Transactions on Computers*, vol.51, pp. 652-668, Jun. 2002.

[126] A. Popoola, "Design Patterns: PubSub Explained," http://abdulapopoola.com/2013/03/12/design-patterns-pub-sub-explained/, [Accessed: 12-08-2014].

[127] XMPP, "XMPP Standard Foundation," http://xmpp.org/about-xmpp/history/, [Accessed: 12-08-2014].

[128] XMPP, "XMPP Pub/Sub extension," *www.xmpp.org/extensions/xep-0060.html*, [Accessed: 12-08-2014].

[129] O. Ozturk, "Introduction to XMPP Protocol and Developing Online Collaboration Applications using Open Source Software and Libraries," *Int Symp. on Collaborative Technologies and System(CTS)*, pp. 21-25, May. 2010.

[130] M. Hapner, R. Burridge, R. Sharma, J. Fialli, K. Stout, "Java Message Service Specification Version 1.1," http://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/atc_comms_services/swim/documentation/media/compliancy/jms-1_1-fr-spec.pdf, Apr. 2002, [Accessed: 13-08-2014].

[131] Ignite Realtime, "Openfire," http://www.igniterealtime.org/projects/openfire/, [Accessed: 13-08-2014].

[132] X. F. Bai, M. Yang, "Design and implementation of web Instant Message System based on XMPP," *Software Engineering and Service Science (ICSESS), 2012 IEEE 3rd International Conference*, *ISBN: 978-1-4673-2007-8*, pp. 83-88, Jun. 2012.

[133] Apache, "ActiveMQ," http://activemq.apache.org/, [Accessed: 15-08-2014].

[134] B. Snyder, D. Bosanac, Rob Davies, "ActiveMQ in Action," *ISBN: 1933988940*, Mar. 2011.

[135] STOMP, "The Simple Text Oriented Messaging Protocol," http://stomp.github.io/, [Accessed: 15-08-2014].

[136] Y. Zhao, I. A. Bond, W. L. Sweatman, "An Android Application for Receiving Notifications of Astrophysical Transient Events," *Astronomy and Computing*, vol. 6, pp. 19-27, Oct. 2014.

[137] I. A. Bond, F. Abe, R. J. Dodd, J. B. Hearnshaw, M. Honda, J. Jugaku, P. M. Kilmartin, A. Marles, K. Masuda, Y. Matsubara, Y. Muraki, T. Nakamura, G. Nankivell, S. Noda, C. Noguchi, K. Ohnishi, N. J. Rattenbury, M. Reid, T. Saito, H. Sato, M. Sekiguchi, J. Skuljan, D. J. Sullivan, T. Sumi, M. Takeuti, Y. Watase, S. Wilkinson, R. Yamada, T. Yanagisawa, P. C. M. Yock, "Real-Time Difference Imaging Analysis of MOA Galactic Bulge Observations During 2000," *Monthly Notices of the Royal Astronomical Society*, vol. 327, pp. 868-880, Jul. 2001.

[138] K. Griest, N. Safizadeh, "The Use of High-Magnification Microlensing Events in Discovering Extrasolar Planets," *The Astrophysical Journal*, vol. 500, pp. 37-50, Jun. 1998.

[139] Oracle, "Oracle WebLogic Server," http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html, [Accessed: 16-08-2014].

[140] OpenJMS Group, "OpenJMS," http://openjms.sourceforge.net/, [Accessed: 16-08-2014].

[141] Progress, "Aurea SonicMQ," https://www.progress.com/products/openedge/solutions/application-integration/aurea-sonic-mq, [Accessed: 16-08-2014].

[142] Apache, "ActiveMQ installation files," http://activemq.apache.org/activemq-5100-release.html, [Accessed: 16-08-2014].

[143] Apache, "ActiveMQ kahadb database", http://activemq.apache.org/kahadb.html, [Accessed: 16-08-2014].

[144] Amazon, "Amazon EC2," http://aws.amazon.com/ec2/, [Accessed: 18-08-2014].

[145] J. Dejun, G. Pierre, C. H. Chi, "EC2 Performance Analysis," *VU University Amsterdam, Tsinghua University Beijing*, http://www.globule.org/publi/EPARPSOA_nfpsla2009.pdf

[146] Twitter, “Twitter Developers Documentation,” https://dev.twitter.com/rest/public, [Accessed: 23-09-2014].

[147] D. Sullivan, “URL Shorteners,” http://searchengineland.com/analysis-which-url-shortening-service-should-you-use-17204, Apr. 2009, [Accessed: 23-09-2014].

[148] R. Allsman, M. N. Santisteban, R. Plante, T. Axelrod, “LSST Alert Management,” *VOEvent Meeting Tucson*, Dec. 2005, [Accessed: 25-09-2014].

[149] Guru, “Top 10 Mobile Phones Operating Systems,” http://www.shoutmeloud.com/top-mobile-os-overview.html, [Accessed: 08-11-2014].

[150] H-G. Schmidt, K. Raddatz, A-D. Schmidt, A. Camtepe, S. Albayrak, “Google Android – A Comprehensive Introduction,” http://www.dai-labor.de/fileadmin/files/publications/GoogleAndroid.pdf, Mar. 16, 2009 [Accessed: 10-11-2014].

[151] Android, “Discovery Android,” http://www.android.com/about/, [Accessed: 11-11-2014].

[152] Egham, “2012, Market Share of Android OS, Gartner Says Worldwide Sales of Mobile Phones Declined 3 Percent in Third Quarter of 2012: Smartphone Sales Increased 47 Percent,” http://www.gartner.com/it/page.jsp?id=2237315, Nov. 2012, [Accessed: 13-11-2014].

[153] H. Yang, “Android Application Package File (APK),” http://www.herongyang.com/Android/APK-What-Is-APK-File-Format.html, [Accessed: 16-11-2014].

[154] B. Truax, 2009. “An iPhone app for Transient Events,” *Hot Wiring the Transient Universe: Infrastructure for Rapid-Response Astronomy*, pp. 60–72, 2010.

[155] A. Gauthier, S. Jacoby, “Transient Events iPhone App,” *Large Synoptic Survey Telescope: E-News*, 2011.

[156] A. Tarantola, “Google’s new cloud messaging system does more for less,” http://www.vogella.com/tutorials/AndroidCloudToDeviceMessaging/article.html, [Accessed: 20-11-2014].

[157] R. W. Keyes, “The Impact of Moore’s Law,” *Solid-State Circuits Society Newsletter, IEEE,* vol.11, pp. 25-27, Sep. 2006.

[158] R. E. Hattachi, J. Erfanian, A. Annunziato, K. Holley, C. Chen, E. Hardouin, L. F. Fei, M. Iwamura, R. Irmer, S. Apetrei, C. Cheeseman, A. Tame, “5G White Paper,” http://www.ngmn.org/uploads/media/NGMN_5G_White_Paper_V1_0.pdf, Feb. 2015, [Accessed: 03-03-2015].

[159] T. Connolly, C. Begg, “Database Systems: A Practical Approach to Design, Implementation and Management. (4th ed.),” *ISBN: 978-0-32121-025-8*, 2004.

[160] E. F. Codd, “Relational Completeness of Data Base Sublanguages,” *Database Systems*, pp. 65-98, 1970.

[161] D. Pinto, “A Very Fast Introduction to DL, DML & DCL,” http://dpinto.cs.buap.mx/bd/VFIntroductionDDL_DML_DCL.pdf, Mar. 2009, [Accessed: 25-04-2015].

[162] S. J. Kleinman, S.O. Kepler, D. Koester, I. Pelisoli, Viviane Pȩcanha, A. Nitta, J. E. S. Costa, J. Krzesinski, P. Dufour, F. -R. Lachapelle, P. Bergeron, C. W. Yip, H. C. Harris, D. J. Eisenstein, L. Althaus, A. Córsico, “SDSS DR7 White Dwarf Catalog,” *The Astrophysical Journal Supplement*, vol. 204, issue. 1, Jan. 2013.

[163] Prof. W. Kriha, “NoSQL Databases,” http://www.christof-strauch.de/NoSQLdbs.pdf, [Accessed: 27-04-2015].

[164] A. K. Zaki, “NOSQL Databases: New Millennium Database for Big Data, Big Users, Cloud Computing and its Security Challenges,” *International Journal of Research in Engineering and Technology*, vol.3, pp. 403-409, May. 2014.

[165] A. Nayak, A. Poriya, D. Poojary, “Type of NoSQL Databases and its Comparison with Relational Databases,” *International Journal of Applied Information System (UAIS)*, vol.5, pp. 16-19, Mar. 2013.

[166] L. Joseji, “11 Open NoSQL Document-Oriented Databases,” http://architects.dzone.com/articles/11-open-NoSQL-document, Jul. 2012, [Accessed: 01-05-2015].

[167] Apache, “Xindice,” https://xml.apache.org/xindice/, [Accessed: 08-06-2014].

[168] K. Staken, “Introduction to Native XML Databases,” http://www.xml.com/pub/a/2001/10/31/nativexmldb.html, Oct. 2001, [Accessed: 03-05-2015].

[169] K. P. Ryan, D. Merriman, E. Horowitz, MongoDB Inc (10 Gen), “MongoDB,” https://www.mongodb.org/, [Accessed: 15-06-2014].

[170] Free Software Foundation GNU Operating System, “GNU Affero General Public License,” http://www.gnu.org/licenses/agpl-3.0.en.html, Nov. 2007, [Accessed: 16-05-2015].

[171] Solid IT, “DB-Engines Ranking,” http://db-engines.com/en/ranking, [Accessed: 16-05-2015].

[172] BSON org, “BSON Document Format,” http://bsonspec.org/, [Accessed: 01-05-2015].

[173] C. G. Wilson, “MongoDB Drivers-Wire Protocol,” http://craiggwilson.com/2013/12/10/mongodb-drivers-wire-protocol-1/, Dec. 2013, [Accessed: 18-06-2014].

[174] MongoDB, “MongoDB Java API Driver,” http://mongodb.github.io/mongo-java-driver/3.0/driver/getting-started/installation-guide/, [Accessed: 15-02-2015].

[175] R. Williams, R. Hanisch, A. Szalay, R. Plante , “Simple Cone Search,” http://www.ivoa.net/documents/latest/ConeSearch.html, Feb. 2008, [Accessed: 01-05-2015].

[176] LSST, “LSST: A New Telescope Concept,” http://www.lsst.org/lsst/public/tour_software, [Accessed: 20-05-2015].

[177] MongoDB, “MongoDB storage,” http://docs.mongodb.org/manual/faq/storage/, [Accessed: 18-05-2015].

[178] MongoDB, “Sharding and MongoDB, release 3.0.3,” http://docs.mongodb.org/master/MongoDB-sharding-guide.pdf, [Accessed: 23-05-2015].

[179] J. Kuhlenkamp, M. Klems, O. Ross, “Benchmarking Scalability and Elasticity of Distributed Database Systems,” *The Proceedings of the VLDB Endowment*, vol.7, pp. 1219-1230, Aug. 2014.

[180] MongoDB, “Sharded Cluster Tutorials,” http://docs.mongodb.org/manual/administration/sharded-clusters/, [Accessed: 23-05-2015].

[181] M. Dominik, K. Horne, A. Allan, N.J. Rattenbury, Y. Tsapras, C. Snodgrass, M. F. Bode, M. J. Burgdorf, S. N. Fraser, E. Kerins, C. J. Mottram, I. A. Steele, R. A. Street, P. J. Wheatley, L. Wyrzykowski, “ARTEMiS (Automated Robotic Terrestrial Exoplanet Microlensing Search) - A possible expert-system based cooperative effort to hunt for planets of Earth mass and below,” *Astronomische Nachrichten*, vol. 329, pp. 248, Mar. 2008.

[182] Jessee, “The Advantages and Disadvantages of XMPP,” http://www.programering.com/a/MjNyEzNwATY.html, Oct. 2014, [Accessed: 13-06-2015].

[183] A. Sinha, S. Sinha, “SILC-A Secured Internet Chat Protocol,” *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, vol. 2, May. 2013.

[184] A. Grould, K. Norne, R. Street, “K2 Microlensing: Free-Floating Planets and Microlens Planet Masses,” http://keplerscience.arc.nasa.gov/K2/docs/Campaigns/C9/K2_ulens_white_paper.pdf, [Accessed: 13-06-2015].

[185] S. Mollerach, E. Roulet, “Gravitational Lensing and Microlensing,” *World Scientific, e-book ISBN: 978-981-4489-35-5*, Jan. 2002.

[186] WebSocket.org, “WebSocket,” https://www.websocket.org/, [Accessed: 13-06-2015].

[187] SDSS, “The Catalog Archive Server Database Management System,” *Computing in Science & Engineering*, vol.10, Feb. 2008.

[188] N. C. Hambly, R. S. Collins, N. J. G. Cross, R. G. Mann, M. A. Read, E. T. W. Sutorius, I. A. Bond, J. Bryant, J. P. Emerson, A. Lawrence, J. M. Stewart, P. M. Williams, A. Adamson, S. Dye, P. Hirst, S. J. Warren, "The WFCAM Science Archive," *Monthly Notices of the Royal Astronomical Society*, vol. 384, pp. 637-662, Nov. 2007.

[189] T. –L. Wu, "An Overview of Present NoSQL Solutions and Features," http://grids.ucs.indiana.edu/ptliupages/publications/An%20Overview%20of%20Present%20NoSQL%20Solutions%20and%20Features%20revised.pdf, [Accessed: 22-06-2015].

[190] A. Salminen, "Introduction to NoSQL," *NoSQL Seminar 2012 @ TUT*, http://www.cs.tut.fi/~tjm/seminars/NoSQL2012/NoSQL-Intro.pdf, [Accessed: 22-06-2015].

[191] S. Gilbert, N. Lynch, "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services," *Newsletter ACM SIGACT News*, vol.33, pp.51-59, Jun. 2002.

[192] CoreFiling, "Online XML Schema Validator," http://www.corefiling.com/opensource/schemaValidate.html.

[193] Edrawsoft, "Edraw Max Pro," https://www.edrawsoft.com/EDrawMax.php.

[194] C. Alcock, R. Allsman, T. Axelrod, D. Bennett, S. Chan, K. Cook, K. Freeman, K. Griest, S. Marshall, S. Perlmutter, B. Peterson, M. Pratt, P. Quinn, A. Rodgers, C. Stubbs, W. Sutherland, C. Alcock, R. Allsman, T. Axelrod, D. Bennett, S. Chan, K. Cook, K. Freeman, K. Griest, S. Marshall, S. Perlmutter, B. Peterson, M. Pratt, P. Quinn, A. Rodgers, C. Stubbs, W. Sutherland, "Probable Gravitational Microlensing Towards the Galactic Bulge," *The Astrophysical Journal*, vol. 445, no. 1, p. 133-139, May. 1995.

[195] G. W. Christie, "Detecting Exoplanets by Gravitational Microlensing using a Small Telescope," *Society for Astronomical Science*, May. 2006.

[196] G. Kin, "Galactic microlensing as a method of detecting massive compact halo objects," *The Astrophysical Journal*, vol. 366, pp. 412-421, Jan. 1991.

[197] B. Paczyński, Apj, 371, L63, 1991.

[198] ActiveMQ, "user group of ActiveMQ," available at: http://activemq.apache.org/users.html, [Accessed: 19-10-2015].

[199] R. Henjes, M. Menth, C. Zepfel, "Throughput Performance of Java Messaging Services Using Sun Java System Message Queue," University of Wuerzburg, available at: http://www.scs-europe.net/services/ecms2006/ecms2006%20pdf/1014-hpc.pdf, [Accessed: 19-10-2015].

[200] Krissoft Solutions, "JMS Performance Comparison," available at: http://www.capitalware.com/dl/docs/krissoft_jms_performance.pdf, [Accessed: 20-10-2015].

[201] ActiveMQ. “JMeter Performance Test,” 2006, available at: http://incubator.apache.org/activemq/jmeter-performance-tests.html , [Accessed: 20-10-2015].

[202] Sonic Software Corporation, “JMS Performance Comparison: SonicMQ(R) vs TIBCO Enterprise(TM) for JMS,” *White Paper*, Nov. 2003, available at: http://www.onwhitepapers.com/redirect.php?wid=4A0D2BDBBE89205241534CCB0AA8ED56, [Accessed: 21-10-2015].

[203] IBM Hursley, “Performance Harness for Java Message Service,” 2005. available at: http://www.alphaworks.ibm.com/tech/perfharness, [Accessed: 23-10-2015]

[204] R.Cattell, “Scalable SQL and NoSQL data stores,” *Newsletter ACM SIGMOD Record*, vol. 39, issue. 4, pp. 12-27, Dec. 2010.

[205] C.Nance, “NOSQL VS RDBMS – Why There Is Room For Both,” *Proceedings of the Southern Association for Information Systems Conference*,” Savannah, GA, USA, Mar. 2010.

[206] A. B. M. Moniruzzaman, S. A. Hossain, “NoSQL Database: New Era of Databases for Big data Analytics - Classification Characteristics and Comparison,” *ArXiv:1307.0191 e-print*, Jun. 2013.

[207] J. M. T. Clarence, S. Aravindh, A. B. Shreeharsha, “Comparative Study of the New Generation,Agile,Scalable,High Performance NOSQL databases,” *International Journal of Computer Applications*, vol. 48, no. 20, 2012.

[208] J.Pokorny, “NoSQL databases: a step to database scalability in web environment,” *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, *ISBN: 978-1-4503-0784-0*, pp. 278-283, 2011.

[209] F. Change, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, “Bigtable:A Distributed Storage System for Structured Data,” ACM Transactions on Computer Systems (TOCS), vol. 26, issue. 2, no. 4, Jun. 2008.

[210] Tutorials Point, “HTML (Hyper Text Markup Language),” available at: http://www.tutorialspoint.com/html/html_tutorial.pdf, [Accessed: 29-10-2015].

[211] H. Y. Lim, “XML (Extensible Markup Language),” available at: http://www.edb.utexas.edu/minliu/multimedia/XML.pdf, [Accessed: 29-10-2015].

[212] ECMA International, “The JSON Data Exchange Format,” Oct, 2013, available at: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, [Accessed: 29-10-2015].

[213] W3C, “Hypertext Transfer Protocol -- HTTP/1.1,” 1999, available at: http://www.w3.org/Protocols/HTTP/1.1/rfc2616.pdf, [Accessed: 29-10-2015].

[214] W3C, “Simple Object Access Protocol,” May. 2000, available at: http://www.immagic.com/eLibrary/ARCHIVES/SUPRSDED/W3C/W000520N.pdf, [Accessed: 29-10-2015].

[215] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” *Doctorate Thesis*, University of California, Irvine, 2000, available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[216] P. Dowler, D. Tody, F. Bonnarel, “Simple Image Access Protocol Version 2.0,” Jul. 2015.

[217] D. Tody, M. Dolensky, J. McDowell, F. Bonnarel, T. Budavari, I. Busko, A. Micol, P. Osuna, J. Salgado, P. Skoda, R. Thompson, F. Valdes, Data Access Layer working group, “Simple Spectral Access Protocol Version 1.1,” Feb. 2012.

[218] R. Williams, R. Hanisch, A. Szalay, R. Plante, “Simple Cone Search Version 1.03,” Feb. 2008.

[219] NASA, “Dying Supergiant Stars Implicated in Hours-long Gamma-Ray Bursts,” Apr. 2013, available at: http://www.nasa.gov/mission_pages/swift/bursts/supergiant-stars.html, [Accessed: 02-11-2015].

[220] Outer Space Central,”Gamma Ray Bursts,” Apr. 2013, available at: http://www.outerspacecentral.com/gamma_ray_page.html, [Accessed: 02-11-2015].

[221] A. Burda, M. Cwiokb, H. Czyrkowskib, R. Dabrowskib, W. Dominikb, M. Grajdaa, M. Husejkoa, M. Jegiera, A. Kalickia, G. Kasprowicza, K. Kierzkowskib, K. Krupskac, K. Kwiecinskac, L. Mankiewiczd, K. Nawrockif, B. Pileckie, L.W. Piotrowskib, K. Pozniaka, R. Romaniuka, R. Salanskia, M. Sokolowskif, D. Szczygiele, G. Wrochnaf, , , , W. Zabolotnya, “Pi of the Sky – all-sky, real-time search for fast optical transients,” *Arxiv e-print*, *astro-ph/0411456v1*, Nov. 2004.

[222] S. D. Barthelmy, T. L. Cline, P. Butterworth, R. M. Kippen, M. S. Briggs, V. Connaughton, G. N. Pendleton, “GRB Coordinates Network (GCN): A status report,” *GAMMA-RAY BURSTS: 5th Huntsville Symposium. AIP Conference Proceedings*, vol. 526, pp. 731-735, 2000.

[223] D. Thompson, “GLAST Large Area Telescope Transient Communications,” available at: http://fermi.gsfc.nasa.gov/ssc/library/fug/070204/DThompson_VO_GCN.pdf, [Accessed: 03-11-2015]

[224] F. Ochsenbein, R. Williams, C. Davenhall, D. Durand, P. Fernique, R. Hanisch, D. Giaretta, T. McGlynn, A. S. A. Wicenec, “VOTable: Tabular Data for the Virtual Observatory,” *Part of the series ESO ASTROPHYSICS SYMPOSIA*, pp. 118-123, Nov. 2004.

[225] S.Kale, T. M. Vijayaraman, P. R. Krishnan, A. Navelkar, H. Hegde, K. D. Balaji, “VOPlot: A Toolkit for Scientific Discovery using VOTables,” *Astronomical Data Analysis Software and Systems XIII*, *ASP Conference Series*, vol. 314, 2004.

[226] M. J. Graham , S. G. Djorgovski, A. Mahabal, C. Donalek, A. Drake, G. Longo, “Data challenges of time domain astronomy,” *Distributed and Parallel Databases on Data Intensive eScience*, vol. 30, issue. 5, pp. 371-384, Oct. 2012.

[227] K. Nilson, “Pushing Data to the Browser with Comet,” Jul. 2008, available at: http://www.developer.com/tech/article.php/3756841/Pushing-Data-to-the-Browser-with-Comet.htm, [Accessed: 07-11-2015].

[228] S. Loreto, P. Saint-Andre, S. Salsano, G. Wilkins, “Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP,” *Internet Engineering Task Force (IETF), ISSN: 2070-1721*, Apr. 2011.

[229] T. Kapyla, I. Niemi, A. Lehtola, “Towards an Accessible Web by Applying PUSH Technology,” VTT Information Technology, Finland, available at: http://ui4all.ics.forth.gr/UI4ALL-98/kapyla.pdf, [Accessed: 07-11-2015].

[230] V. Kumar, “Fundamentals of Pervasive Information Management Systems,” *eBook : Document*, 2013.

[231] Hermans, “A Prominent Change Driver: Push Technology,” available at: http://www.hermans.org/agents2/ch2.htm, [Accessed: 09-11-2015].

[232] D. C. Schmidt, C. O'Ryan, “Patterns and Performance of Distributed Real-time and Embedded Publisher/Subscriber Architectures,” *Journal of Systems and Software - Special issue on: Software architecture - Engineering quality attributes*, vol. 66, issue. 3, pp. 213-223, Jun. 2003.

[233] SonicMQ, “Progress Software Corporation,” available at: https://docs.oracle.com/cd/E18867_01/SRE/GettingStarted.pdf, [Accessed: 10-11-2015].

[234] G. Fox, S. Pallickara, “JMS Compliance in the Narada Event Brokering System,” Indiana University, available at: http://www.naradabrokering.org/papers/JMSSupportInNaradaBrokering.pdf, [Accessed: 11-11-2015].

[235] M. Pohja, “Server Push for Web Application via Instant Messaging,” *Journal of Web Engineering*, vol. 9, issue. 3, pp. 227-242, Sep. 2010.

[236] X. Bai. M. Yang, “Design and implementation of web Instant Message System based on XMPP,” Software Engineering and Service Science (ICSESS), IEEE 3rd International Conference, pp. 83-88, Jun. 2012.

[237] M. Laine, K. Säilä, “Performance Evaluation of XMPP on the Web,” *Technical report*, Apr. 2012, available at: http://media.tkk.fi/webservices/personnel/markku_laine/performance_evaluation_of_xmpp_on_the_web.pdf, [Accessed: 11-11-2015].

[238] Willy Farrell, "Introducing the Java Message Service," *IBM technical report*, available at: http://www.digilife.be/quickreferences/pt/introducing%20the%20java%20message%20service.pdf, [Accessed: 11-11-2015].

[239] D. Jun, Z. X. Min, "Design and Implementation of an Asynchronous Message Bus Based on ActiveMQ," *Computer systems & applications*, Aug. 2008.

[240] S. Chen, P. Greenfield, "QoS Evaluation of JMS: An Empirical Approach," *System Sciences, Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Jan. 2004.

[241] A. Foster, "Messaging Technologies for the Industrial Internet and the Internet of Things," *PrismTech Whitepaper*, May. 2015.

[242] Apollo, "OpenWire Protocol Manual," available at: http://activemq.apache.org/apollo/documentation/openwire-manual.html, [Accessed: 12-11-2015].

[243] JBoss Community, "HornetQ," available at: http://hornetq.jboss.org/, [Accessed:13-11-2015].

[244] Oracle, "Oracle Communications Instant Messaging Server: Real-Time Collaboration for Service Providers and the Extended Enterpris," *Oracle White Paper*, Jul. 2014.

[245] D. R. Nelson, "Method and Apparatus for providing automated notification to a customer of a real time notification system," *US Patents US6496568 B1*, Apr. 1999.

[246] M. A. Nawi, N. S. Haron, M. H. Hasan, "Context-aware instant messenger with integrated scheduling planner," *Computer & Information Science (ICCIS), 2012 International Conference*, vol. 2, pp. 900-907, Jun. 2012.

[247] S. K. Change, X. Li, R. Villamarin, D. Lyker, C. Bryant, "The Design and Implementation of the Chronobot/Virtual Classroom(CVC)," *System,International Conference on Distributed Multimedia Systems*, Aug. 2006.

[248] S. R. A. Aziz, M. Ibrahim, M. S. Sauti, "The Exploitation of Instant Messaging to Monitor Computer Networks Using XMPP: A Study Focuses on School Computer Labs," *Journal of Advanced Management Science*, vol. 4, no. 3, May. 2015.

[249] A. Delis, N. Roussopoulos, "Management of updates in the enhanced client-server DBMS," *Distributed Computing Systems*, *In Proceedings of the 14th International Conference*, pp. 326-335, Jun. 1994.

[250] V. Kanitkar, A. Delis, "Real-Time Client-Server Push Strategies: Specification and Evaluation," *Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*, pp. 179-188, Jun. 1999.

[251] L. Vargas, J. Bacon, K. Moody, “Integrating Databases with Publish/Subscribe,” Distributed Event-Based Systems (DEBS) Workshops, 25th IEEE International Conference, pp. 392-397, Jun. 2005.

[252] M. Gusev, S. Ristov, G. Velkoski, A. Guseva, P. Gushev, “Scalable Architecture of Alert Notification as a Service,” *Information Society (i-Society), International Conference*, pp. 80-85, Nov. 2014.

[253] P. Brett, R. Knauerhase, M. Bowman, R. Adams, A. Nataraj, J. Sedayao, M. Spindel, A Shared Global Event Propagation System to Enable Next Generation Distributed Services, *WORLDS' 04 Technical Program*, 2004.

[254] I. A. Bond, W. Lin, W. Sweatman, “Delivering Machine Readable Microlensing Data Across the Internet,” *Manchester Microlensing Conference: 12th International Conference and ANGLES Microlensing Workshop*, pp. 047, Jan. 2008.

[255] M. Girdley, R. Woollen, L. S. Emerson, “J2EE Applications and BEA WebLogic Server,” *ISBN-10:0-13-091111-9*, 2002.

[256] Oracle, “Java Authentication and Authorization Service (JAAS) Reference Guide,” available at: http://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html, [Accessed: 17-11-2015]

[257] RED HAT JBOSS FUSE, “JAAS JDBC Login Module,” available at: https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_Fuse/6.0/html/Security_Guide/files/JAASAuth-JDBCLoginModule.html, [Accessed: 18-11-2015]

[258] Oracle, “JDBC Overview,” available at: http://www.oracle.com/technetwork/java/overview-141217.html, [Accessed: 18-11-2015]

[259] Oracle, “Statements and Prepared Statements,” available at: http://www.oracle.com/technetwork/testcontent/jdbc-ch5-131209.pdf, [Accessed: 18-11-2015].

[260] R. Korra'ti, “HowIT Works: How Simple is SMTP,” *TechNet Magazine*, Nov. 2005, available at: https://technet.microsoft.com/en-us/magazine/2005.11.howitworkssmtp.aspx, [Accessed: 20-11-2015].

[261] Omnisecu, “Simple Mail Transfer Protocol (SMTP)-How SMTP works?,” available at: http://www.omnisecu.com/tcpip/smtp-simple-mail-transfer-protocol-how-smtp-works.php, [Accessed: 20-11-2015].

[262] M. Hapner, R. Burridge, R. Sharma, Sun Microsystems, “Java™ Message Service version 1.0.2,” Nov, 1999, available at: https://docs.oracle.com/cd/E19957-01/816-5904-10/816-5904-10.pdf, [Accessed: 20-11-2015].

[263] Android, "AsyncTask," available at: http://developer.android.com/reference/android/os/AsyncTask.html, [Accessed: 21-11-2015].

[264] B. G. Park, "KMTNet-Korean Microlensing Telescope Network," available at: http://www.astronomy.ohio-state.edu/~microfun/MF1/Talks/Park_KMTNet.pdf, [Accessed: 21-11-2015].

[265] W. Reinhardt, S. Schmid, E. Galice, "Performance investigation of selected SQL and NoSQL databases," *AGILE Conference*, 2015.

[266] C. J. M. Tauro, S. Aravindh, A. B. Shreeharsha, "Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases," *International Journal of Computer Applications*, vol. 48, no. 20, Jun. 2012.

[267] A. Oussous, F. Z. Benjelloun, A. A. Lahcen, S. Belfkih, "Comparison and Classification of NOSQL Databases for Big Data," *Conference: International conference on Big Data, Cloud and Applications*, *Arxiv eprint: 1307.0191*, Jun. 2013.

[268] V. Abramova, J. Bernardino, P. Furtado, "Which NoSQL Database? A Performance Overview," *Open Journal of Databases (OJDB)*, *ISSN 2199-3459*, vol. 1, issue. 2, 2014

[269] K. Griest, N. Safizadeh, "The Use of High Magnification Microlensing Events in Discovering Extrasolar Planets," *The Astrophysical Journal*, vol. 500, pp. 37-50, Jun. 1998.

[270] Y. Huang, T. Luo, "NoSQL Database: A Scalable, Availability, High Performance Storage for Big Data," *Pervasive Computing and the Networked World*, vol. 8351, pp. 172-183, 2014

[271] MongoDB, "2D Search," available at: https://docs.mongodb.org/manual/tutorial/query-a-2d-index/, [Accessed: 28-11-2015]

[272] N. Slater, "Best Practices for Migrating from RDBMS to Amazon DynamoDB – Leverage the Power of NoSQL for Suitable Workloads," *Amazon Whitepaper*, Mar. 2015.

[273] A. Salehnia, "Comparisons of Relational Databases with Big Data: a Teaching Approach," South Dakota State University, *Technical Report*, available at: https://www.asee.org/documents/zones/zone3/2015/Comparisons-of-Relational-Databases-with-Big-Data-a-Teaching-Approach.pdf, [Accessed: 01-12-2015].

[274] R. P. Padhy, M. R. Patra and S. C. Satapathy, "RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's," *International Journal of Advanced Engineering Science and Technologies*, vol. 11, no. 1, pp. 15-30, 2011.

[275] M. A. Mohamed, O. G. Altrafi, M. O. Ismail, "Relational vs. NoSQL Databases: A Survey," *International Journal of Computer and Information Technology*, vol. 03, issue. 03, May. 2014.

[276] K. Chitra, B. JeevaRani, "Study on Basically Available, Scalable and Eventually Consistent NOSQL Databases," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, issue. 7, Jul. 2013.

[277] J. Gray, A. Szalay, A. Thakar, P. Z. Zunszt, T. Malik, J. Raddick, C. Stoughton, J. vandenBerg, "The SDSS SkyServer – Public Access to the Sloan Digital Sky Server Data," *TechReport*, *MSR-TR-2001-104*, Nov. 2001.

[278] Microsoft, "Microsoft SQL server," available at: http://www.microsoft.com/en-us/server-cloud/products/sql-server/, [Accessed: 03-12-2015]

[279] A. R. Thakar, A. Szalay, G. Fekete, J. Gray, "The Catalog Archive Server Database Management System," *Computing in Science & Engineering*, vol. 10, issue. 01, pp. 30-37, Jan. 2008.

[280] R. A. Power, "Large Catalogue Query Performance in Relational Databases," *Publication of the Astronomical Society of Australia*, vol. 24, issue. 02, pp. 13-20, May. 2007.

[281] MySQL, "MySQL database system," available at: https://www.mysql.com/, [Accessed: 03-12-2015]

[282] Oracle, "Oracle Database," available at: http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html, [Accessed: 03-12-2015].

[283] L. Nicastro, G. Calderone, "Concepts for astronomical data accessibility and analysis via relational database," *Proceedings of the 363. WE-Heraeus Seminar on: Neutron Stars and Pulsars (Posters and contributed talks)*, pp. 14-19, May. 2006.

[284] R. J. Brunner, J. Gray, P. Kunszt, D. Slutz, A. S. Szalay, A. Thakar, "Designing and Mining Multiterabyte Astronomy Archives: The Sloan Digital Sky Survey," *Proc. Special Interest Group Management of Data Conf. (SIGMOD), ACM Press*, pp. 451–462, 2000.

[285] A. Thakar, A. Szalay, P. Kunszt, J. Gary, "Migrating a Multiterabyte Archive from Object to Relational Databases," *Computing in Science and Engineering*, vol. 5, issue. 5, pp. 16-29, Sep. 2003.

[286] J. Letkowski, "Doing database design with MySQL," *Journal of Technology Research*, vol 6, Dec. 2014.

[287] D. I. Inan, R. Juita, "Analysis and Design Complex and Large Database using MySQL Workbench," *International Journal of Computer Science & Information Technology (IJCSIT)*, vol. 3, no. 5, Oct. 2011.

[288] P. A. Larson, J. Goldstein, J. Zhou, "MTCache: Transparent Mid-Tier Database Caching in SQL Server," *Data Engineering, Proceedings. 20th International Conference on*, pp. 177-188, Mar. 2004.

[289] D. J. Abadi, “Data management in the cloud: Limitations and opportunities,” *IEEE Data Engineering*, vol. 32, no. 1, pp. 21-27, Mar. 2009.

[290] F. Change, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, **“**Bigtable: A Distributed Storage System for Structured Data,**”** *ACM Transactions on Computer Systems (TOCS)*, vol. 26, issue. 2, Jun. 2008.

[291] Couchbase, “Couchbase server,” available at: http://couchbase.com/press-releases/couchbase-survey-shows-accelerated-adoption-NoSQL-2012, [Accessed: 05-12-2015].

[292] Y. Li, S. Manoharn, “A performance comparison of SQL and NoSQL databases,” *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*, pp. 15-19, Aug. 2013.

[293] S. S. Nyati, S. Pawar, R. Ingle, “Performance evaluation of unstructured NOSQL data over distributed framework,” *Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1623-1627, Aug. 2013.

[294] V. Sharma, M. Dave, “SQL and NoSQL Databases,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, issue. 8, Aug. 2012.

[295] R. Hecht, S. Jablonski, “NoSQL evaluation: A use case oriented survey,” *Cloud and Service Computing (CSC)*, pp. 336-341, Dec. 2011.

[296] A. Boicea, F. Radulescu, L. I. Agapin, “MongoDB vs Oracle – database comparison,” *Third International Conference on Emerging Intelligent Data and Web Technologies*, pp. 330-335, Sep. 2012.

[297] T. Sumi, K. Kamiya, A. Udalski, D.P. Bennett, I.A. Bond, F. Abe, C.S. Botzler, A. Fukui, K. Furusawa, J.B. Hearnshaw, Y. Itow, P.M. Kilmartin, A. Korpela, W. Lin, C.H. Ling, K. Masuda, Y. Matsubara, N. Miyake, M. Motomura, Y. Muraki, M. Nagaya, S. Nakamura, K. Ohnishi, T. Okumura, Y.C. Perrott, N. Rattenbury, To. Saito, T. Sako, D.J. Sullivan, W.L. Sweatman, P.J. Tristram, P.C.M. Yock, M.K. Szymanski, M. Kubiak, G. Pietrzynski, R. Poleski, I. Soszynski, L. Wyrzykowski, K. Ulaczyk, “Unbound or Distant Planetary Mass Population Detected by Gravitational Microlensing,” *Nature*, vol. 473, issue. 7347, pp. 349-352, 2011.