University of Sheffield

Analysis of log file data produced by a distributed system

Bingyi Wang/ Mwiza Kunda/ Yue Zheng/ Yuliang Li/ Ziling Li

Supervisor: Kirill Bogdanov/ WANdisco International Ltd.

A report submitted for COM6911 Industrial Team Project

in the

Department of Computer Science

May 13, 2019

Contents

Chapter 1

Introduction

A distributed system often faces failure as it features a number of nodes and components. Long running applications and intensive workloads are common in such systems and can result in various failures, including network or memory based failures. WanDisco Fusion is a multi-region and multi-cloud software platform that offers unparalleled, omni-directional replication technology which guarantees user's data to be available, accurate and protected, it can ensure data consistency and accessibility in any environment.

It is therefore of interest to identify any potential indicators of failure of WANdisco Fusion so that appropriate actions can be put in place prior to failure to maintain the accessibility and coherence of user data, in all regions. However, the distributed architecture of WANdisco Fusion results in a high rate of events being logged at the same time by nodes in different regions/zones, making it difficult to extract structure and correlation from log data that can be used to predict any form of failure.

In this project the following is accomplished:

1. A suitable data schema is created for the logs originating from WANdisco's Fusion software.
2. Features are identified that show a strong correlation with the symptoms of an out of memory (OOM) node failure.
3. OOM failure prediction models using Logistic Regression and Support Vector Machines are developed that show promise in predicting OOM failure within a given timeframe.
4. An application of Principal Component Analysis (PCA) is used to showcase a method for determining the "Sickness" of a Fusion Server.

The rest of this report will be organised as follows: In Chapter 2, a literature review is conducted in related areas such as time series analysis for failure prediction; In Chapter 3, the architecture and log data of WANdisco Fusion is detailed; log based feature engineering is presented in Chapter 4; Chapter 5 introduces the failure prediction model and Chapter 6 reports the observed results of the developed model. Finally, a discussion is given in Chapter 7 and a conclusion in Chapter 8.

Chapter 2

Background

Fault tolerant requirements for large scale distributed systems have emerged, Lavinia et al. (2011) believes it is necessary to detect failure to make systems work despite possible faults occurring.

Various methods can be used to predict failure in distributed systems, in our studies, we will focus on approaches to analysis log information from WANdisco cooperation.

In some cases, traditional machine learning techniques can accurately predict the failures. Samak et al. (2012) presents a Naive Bayes classifier to analyze the failure and success rates of jobs on systems. In order to develop effective fault tolerant strategies when the size and complexity of systems rapidly grow, Liang et al. (2007) compares nearest neighbor classifier, RIP-PER(a rule based classifier), and SVM by comparing their failure prediction performance on event logs over an extensive period from IBM BlueGene/L, based on the result, they suggest that the nearest neighbor approach is the most efficient one.

Another popular choice is time series analysis, it observes the past data to determine whether failures will occur in the future. Time series analysis can be helpful to detect anomaly in many fields, for example, Rodriguez & de los Mozos (2010) apply time series analysis to study the past patterns to obtain future decisions by building model on traffic log, and this approach improved network security. For distributed system failure detection, time series analysis can also be helpful. Montgomery et al. (2015) states that many forecasting problems are based on the dataset involves timestamp, time series plots of data could be visually inspected for recognizable patterns such as trends. Time-series analysis is used in this project since many system messages are separated allocated in different log files and timestamp is a good feature to link all log information together.

But these methods mentioned above are not suitable to predict failure for WANdisco Fusion because of the log complexity, so we have done more research beyond the time series analysis.

Preprocessing is really important for log analysis in huge size systems because there are too many important and relevant information related to failure. To detect failure patterns in messages using unlabelled data, Gurumdimma et al. (2015) propose a preprocessing method which will remove redundant data first and then execute clustering algorithm on the resulting logs. Zheng et al. (2009) also presents a log preprocessing method to predict failure, they use event filtering to remove redundant records, and finally combine correlated events by using causality-related filtering, this preprocessing approach can preserve more failure patterns for failure analysis, thereby improving failure prediction.

Based on these research, we will also apply preprocessing methods before building model, based on our studies, the main methods can be grouped into two parts: k-means and PCA.

Clustering time series sequence is important for our project, but most algorithms are not suitable to handle it. Vlachos et al. (2003) argues that many traditional machine learning algorithms do not work well for clustering time series, so he performs clustering of time-series by using k-means algorithm.

For k-means algorithm, Jain (2010) concludes that clustering is a good method to organize data into groups and K-means is a widely used clustering algorithm, it can identify the closest cluster centers for all the data points and then finish clustering, he also states the procedures of k-means, the input is a set $\mathbf{X} = \{x_1, x_2, ..., x_n\}$ of n data points and number of clusters $\mathbf{k}$, for a set $\mathbf{C} = \{c_1, c_2, ..., c_k\}$ of cluster centers, we can define the sum of squared error(SSE) as

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} \|x_i - \mu_k\|^2$$

the goal of k-means is to minimize the sum of the squared error over all $\mathbf{K}$ clusters.

Hautamaki et al. (2008) compares different time-series clustering methods and find out that the prototype with clustering followed by k-means provides the best accuracy, his k-means algorithm consists of partition and prototype, each time-series is mapped to its nearest prototype time-series in partition step, then a new prototype is computed for each subset, finally iterates these two steps until convergence. For time series data analysis, Huang et al. (2016) also introduces a new k-means type model which is able to automatically weight the time stamps according to the importance of a time span in the clustering process, they build time series clustering model by developing a new objective function firstly and then build corresponding iterative clustering rules on the function.

Besides, PCA index construction is helpful to find out failure issues of WANdisco system. Donaubauer et al. (2016) states PCA can assign weights to different indicators within an aggregate index and it offers one possible approach to condense highly correlated indicators and build an index of overall infrastructure. Abeyasekera (2003) concludes that a new set of variables is created as linear combinations of the original set in PCA, the linear combination which explains the maximum amount of variation is called the first principal component, and further components are then created sequentially, all components are independent of the previous ones, he also suggests that appropriateness of variables should be included in the calculation of the index in relation to the objectives of the analysis.

For PCA algorithm, Tipping & Bishop (1999) derivates the mathematical procedures, for the given data $\{\mathbf{y}\}$, PCA finds orthogonal directions defined by a projection $\mathbf{U}$ capturing the maximum variance in data. The data is $\mathbf{x} = \mathbf{U}^T\mathbf{y}$ in PCA representation, maximising the variance is equivalent to minimise the reconstruction error. By using Lagrange multipliers, we can get,

$$\mathbf{L}(\mu_\mathbf{1}, \lambda) = \mu_\mathbf{1}{}^T \mathbf{S} \mu_\mathbf{1} + \lambda_1 (1 - \mu_\mathbf{1}{}^T \mu_\mathbf{1})$$

gradient with respect to $\boldsymbol{\mu_1}$we can get,

$$\frac{d\mathbf{L}(\boldsymbol{\mu_1}, \lambda_1)}{d\mu_1} = 2\mathbf{S}\boldsymbol{\mu_1} - 2\boldsymbol{\lambda_1}\boldsymbol{\mu_1}$$

rearrange to form:

$$\mathbf{S}\boldsymbol{\mu_1} = \boldsymbol{\lambda_1}\boldsymbol{\mu_1}$$

further directions that are orthogonal (uncorrelated) to the first can also be shown to be eigenvectors of the covariance.

After preprocessing, we need some classification models to predict failure, we did research about classification algorithm.

Mayr et al. (2014) proposed the basic idea of boosting algorithms is to iteratively apply simple classifiers and to combine their solutions to obtain a better prediction result. Schapire (2003) argues that boosting is a general method for improving the accuracy of any given learning algorithm. Adewale et al. (2010) tests functional gradient descent boosting for the classification of correlated binary data with high-dimensional predictors, he states that the variants of boosting could address classification issues.

Logistic Regression can be used as a great classifier in this situation. Bewick et al. (2005) concludes that Logistic regression provides a method for modelling a binary response variable, which takes values 1 and 0 which can be regarded as labels. Komarek (2004) also states that logistic regression is a capable and fast tool for data mining and binary classification. Predicting failure is actually a binary classification problem, so we also use logistic regression to figure out failure.

Hosmer Jr et al. (2013) demonstrates the mathematical points about logistic regression, based on a standard regression, take $f(x) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$, a logistic regression should be

$$\log \frac{\pi}{(1-\pi)} = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$$

by setting a inverse link function $g^{-1}(\pi) = \mathbf{w}^\top \phi(\mathbf{x})$, then use π to represent it as

$$\pi(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp\left(-\mathbf{w}^\top \phi(\mathbf{x})\right)}$$

next, apply this to objective function:

$$E(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log g\left(\mathbf{w}^\top \boldsymbol{\phi}\left(\mathbf{x}_i\right)\right) - \sum_{i=1}^{n} (1 - y_i) \log\left(1 - g\left(\mathbf{w}^\top \boldsymbol{\phi}\left(\mathbf{x}_i\right)\right)\right)$$

then minimize this objective by differentiating with respect to the parameters of $\pi(\mathbf{x}, \mathbf{w})$.

SVM is widely used for classification. Fulp et al. (2008) introduces a failure prediction method using SVM based on the log information, they took advantage of the sequential nature of log messages and determines which sequence of messages can be used to forecast failure, the approach will use a sub-sequence of messages to predict the likelihood of failure, and finally SVM associates the messages to classes of failed and non-failed. But SVM may not always be helpful in all situations, Fronza et al. (2013) found that normal SVM can only find out non-failures, so they use weighted SVM instead of normal type to classify both non-failures and failures correctly.

Suykens & Vandewalle (1999) demonstrates the procedures of SVM, he proposes that the SVM classifier is constructed of the form:

$$y(x) = \operatorname{sign}\left[\sum_{k=1}^{N} \alpha_k y_k \psi\left(x, x_k\right) + b\right]$$

where α_k are positive real constants and b is a real constant, and $\psi\left(x, x_k\right) = x_k^T x$ for Linear SVM, $\psi\left(x, x_k\right) = \exp\left\{-\left\|x - x_k\right\|_2^2 / \sigma^2\right\}$ for Radial Basis Function(RBF) SVM, where σ is a constant, the classifier is constructed as

$$y_k\left[w^T \varphi\left(x_k\right) + b\right] \geq 1 - \xi_k, \quad k = 1, \ldots, N$$

where $\xi_k \geq 0, \quad k = 1, \ldots, N$, and $\varphi(\cdot)$ is a nonlinear function which can map the input space into higher dimensional space.

There are still some important issues which could be helpful for detecting failure of WANdisco Fusion. Garbage collection analysis is also really important to judge whether the WANdisco system is healthy, G1 Java Garbage Collection logging is an important part of Fusion Server logs, it gives historical view of Java Heap size and duration of GC events. Detlefs et al. (2004) states that garbage first is a server-style garbage collector, its heap operations are performed concurrently with mutation in order to prevent interruptions to heap or live-data size. In order to analyze GC performance, Kejariwal (2013) presents a tool called Shrek, Shrek facilitates analysis of GC logs of Java applications deployed across hundreds of nodes, it can report the GC statistics for each node. Shrek can be used to detect memory leaks and predict trends of heap usage, it also supports analytics such as time series analysis of GC performance metrics to determine which node is bad and support visualization of information such as promotion rate from the young generation to the old generation.

Memory-related problems are common in many systems, this is especially true for long-running server-style systems where small problems such as memory leaks or excessive storage overhead can result in problems over the running system. In order to avoid or fix such problems, engineers need to understand how the system uses memory. Reiss (2009) designed a visualization tool to let the programmer both understand overall memory behavior and address these and other memory-related problems

Another important concept of WANdisco systems is state machine, distributed software is often structured in terms of clients and services. Schneider (1990) defines that a state machine consists of state variables and commands which is implemented by a deterministic program, a client of the state machine makes a request, which names a state machine, to execute a command. He also argues that state machine approach is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas.

Chapter 3

Overview

3.1 WANdisco Fusion Overview

The central aim of WANdisco's Fusion software is to ensure that data, where replicated between different regions is available and is coherent when accessed in each individual zone. To understand the following chapters, knowledge of the WANdisco system is necessary. WANdisco's Fusion system is accomplished through communication between representatives (specifically servers) in each zone. The architecture of WANdisco's Fusion software is given below in Figure 3.1. The systems main components are the Fusion client, IHC server, Fusion UI server (not shown in Figure 3.1) and Fusion server (source and destination zones). The specific details of this communication process is given below in Section 3.1.2. The general behaviour of the components in the Fusion software is detailed below.

3.1.1 System components

WANdisco (2019) provides the basic information of their system. Figure 3.1 below shows a source zone (right) and a destination zone (left). The communication that allows consistency in data between zones occurs between the two Fusion Servers. In general, there may be any number of Fusion servers in each zone. Fusion servers (Nodes) in different zones are linked together by "replicated paths"; file system paths where data is to be replicated between zones.

1. The IHC server's purpose is to transfer data from local Hadoop Distributed File System (HDFS) to destination fusion servers. It also acts as local HDFS client for the purpose of within zone data movement (intra zone replication).

2. The Fusion UI server interfaces to HDFS for displaying file browsers, acting as the visual representation of the Fusion software for the end user.

3. Fusion server is where the *Fusion Kernel* code is executed ("communication" program, see Section 3.1.2 below). The Fusion server deals only with HDFS write requests on replicated paths. If there is more than one Fusion server in a replicated zone, then only one Fusion node in the zone is responsible for *Proposing* (requesting changes to replicated data) (see label 2 in Figure 3.1) to other Fusion nodes and executing requests for a replicated path. This node is termed a *writer* node.

4. In the source zone, The Fusion Server works as proxy, proxying requests to the local Name Node. It is not involved in the data writing pipeline from the HDFS client to the data nodes (see labels 1 and 5), but it is responsible for obtaining agreement from other Fusion servers for the sequencing of the write requests, and once obtained, for the submitting of the write requests to the name node. This is contrasted with the Fusion server in the remote zone (left) which is directly involved in the data writing pipeline (see label 12).

5. For a *writer* node, execution of requests involves acting as a Hadoop client to the local HDFS file system (see label 10), contacting the originating zone's IHC server(s) to pull the data from the HDFS file system of the source zone (see label 9) and writing the data to the local HDFS file system.

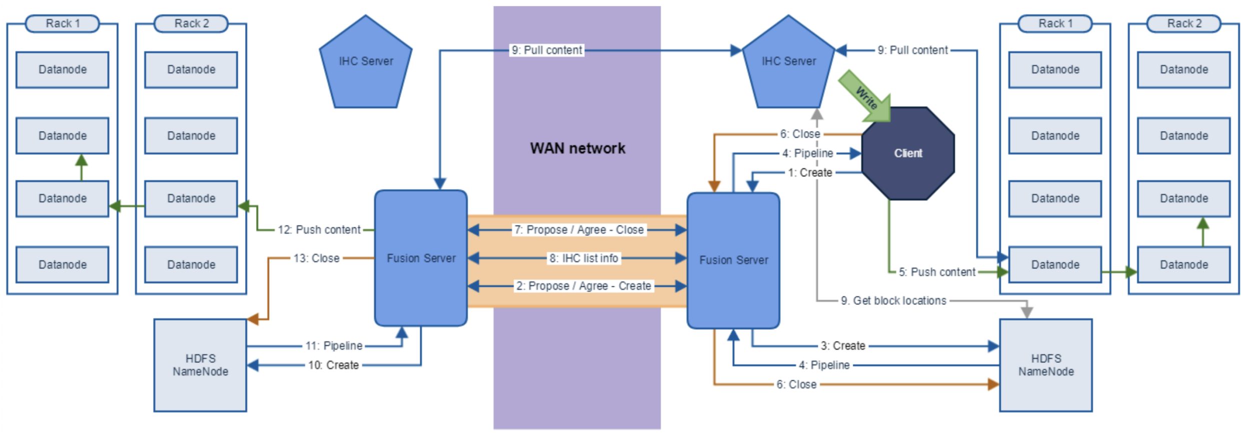


Figure 3.1: *Architecture of WANdisco Fusion*

3.1.2 State machines

Communication between Fusion servers is implemented by the *Fusion kernel* via Deterministic Finite State Machines (FSM). A FSM is a state based transition system that consists of a finite set of states, beginning with an initial state and a set of conditions for each state that determine the transition occuring between a given state and the next state in the system. A deterministic Finite State Machine, or simply Deterministic State Machine (DSM) is a restricted Finite State Machine. For any state in the system, a given input yields exactly the same transition.

To understand how state machines are used in the communication between Fusion Servers, some detail of the Paxos Algorithm is needed. Cocagne (2016) states taht the Paxos algorithm can be encapsulated as a mechanism for achieving "consensus on a single value over unreliable communication channels". That is, given a set of entities, with each entity capable of proposing a message to be agreed upon by the entire set, the Paxos algorithm provides a manner in which agreement can be achieved. Paxos is derived from a simple agreement system. Given a proposal ψ with id number ν by entity $i := 1, \ldots, N$ and a set of entities of size N involved in the agreement process, the proposal ψ is accepted if the majority of the N entities are in agreement of the proposal by entity i. Paxos exceeds this simple system by adding further steps to ensure that a consensus among a set of entities can be achieved even when some of them are unable to participate due to events such as node failure.

In order to be able to replicate transactions for a HDFS path from a source zone to a destination zone, a DSM for that path is needed on each Fusion server in the source zone and destination zone. Operations are coordinated between source and destination zones' fusion servers via messages(state machines) called *proposals*. Different proposals for the same replicated path (DSM) can therefore be viewed as sub state machines. Proposals can be used for the election of *writer* nodes (termed InitialWriterProposal) as well as HDFS operations via RequestProposals (e.g RenameRequest for the renaming of files or DeleteRequest). Every proposal has a Paxos based key (ν) called a Global State Number (GSN) and goes through various states, and this can be seen in the fusion-server log file (more below in Section 3.2.1). The GSN is incremented in units so that successive proposals towards a specific replicated path (DSM) differ by 1.

The general type of state machine in the *Fusion kernel* is as follows:

[Proposing] $\rightarrow$ Observing $\rightarrow$ Delivering $\rightarrow$ Received $\rightarrow$ [Submitting] $\rightarrow$ [Started Executing] $\rightarrow$ Completed

Brackets here signify the states that are only accessible to *writer* nodes. The *Proposing* state is the initiation of a request, and it is here where the proposal for the voting (paxos agreement) of the GSN is initiated. Once agreement has taken place, each node will log that it knows about the proposal (state *Observing*). The state machine is then advanced to the *Received* state to log knowledge of the specific state machine. For the writer node, the state machine is advanced to the *Submitting* and *Started executing* states to execute the suggested proposal (eg Rename of a file). Proposals to be executed are scheduled in a matrix called an Adjacency matrix. The final state of the state machine is completed which is logged for all nodes, be the outcome of the state machine a failure or a success. With state machines such as the above, Fusion servers are able to agree/disagree upon changes to replicated paths as well as log processes of replicated paths.

3.2 WANdisco System outputs (raw data)

3.2.1 Fusion server log

The logging of FSM introduced above generally follows the following structure:

1. Timestamp: recorded time of the log entry
2. Level: There are 5 log levels in the system: PANIC, SEVERE, ERROR, WARNING and INFO. PANIC will always result in shutdown of the fusion server, SEVERE is bad and will likely lead to a shutdown/ system that is no longer functioning correctly. ERROR flags unexpected issues in the system, WARNING highlights normal failures and INFO is sometimes used to capture interesting errors.
3. State: state of the current proposal e.g *Observing*.
4. Node ID: fusion server node ID.
5. DSM ID: unique identifier of a DSM instance (replicated path shared between Fusion Servers).
6. GSN: Paxos based agreement key. This identifies a specific agreed state machine. State machines can be tracked using this key, but to uniquely identify the state machine, the DSM

ID is also needed.

7. Request type: Type of proposal e.g InitialWriterProposal or Create Request.
8. Proposer: Identifier of the proposing node.
9. Request name: Type of request e.g RenameRequest of a specific file.
10. Request ID: Another unique identifier of a specific FSM.

3.2.2 Garbage collection log

Basic background

Automatic garbage collection is the process of handling heap memory, identifying which objects are in use and which are not used, and then processing the unused objects for memory management. Java garbage collection is the process by which Java programs perform automatic memory management. Garbage Collector is a daemon thread. A daemon thread runs behind the application. It is started by JVM(Java virtual machine), The thread stops when all non-daemon threads stop.

The JVM controls the Garbage Collector; it decides when to run the Garbage Collector. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory. When JVM starts up, it creates a heap area which is known as runtime data area which is limited. All the objects are stored in this area and the objects that are no longer in use will be removed because this area is limited and it is necessary to manage it efficiently. The process of removing unused objects from heap memory is known as Garbage collection and this is a part of memory management in Java. If the memory is not enough when threads are running and creating new objects in the running process, JVM will apply the garbage collection in order to reclaim memory for new allocation.

The task of garbage collection in the JVM is to automatically determine what memory is no longer being used by a Java application and to recycle this memory for other uses. GC makes Java memory efficient because unreferenced objects are automatically removed from the heap memory, but if overall memory utilization is increasing continuously despite garbage collection, there is a memory leak, which will inevitably lead to out of memory error.

WANdisco garbage collection

According to WANdisco Fusion logs, it records the version of java they used in gc.logs, the recording of java version is: Java HotSpot(TM) 64-Bit Server VM (25.151-b12) for linux-amd64 JRE (1.8.0 151-b12), and they actually use G1 garbage collection. We can compare the sizes for each generation before and after the garbage collection to find the anomaly. For example, after garbage collection, the eden space should be at or close to zero, while the survivor space gets bigger, and the eden space is also larger than the survivor space.

G1(garbage first) garbage collection is actually generational collector, it splits the heap into young and old generations in order to predict soft target time and keep consistent throughput of application. Garbage first collector collects areas where the amount of live data is least, this process is actually the meaning of garbage first, and the collector will evacuate live data into new regions. A region is a block of allocated space which can hold objects of generations. All objects which are new allocated will try to allocate space in eden, and the live objects after young garbage collection will be copied to survivor area until this object being collected or the age of object achieve the specific value and change to old generation.

JVM will allocate regions when it starts, the empty regions will be added into a unordered linked list which is also known as the “free list”. When the object begins production, a region is allocated as TLAB(thread local allocation buffer) in the free list, and it can use compare and swap to synchronize. Then objects will be allocated in the local buffer, when this non continuous region is full, the new region will be chosen until all space in the cumulative eden region is filled. When all regions are filled, it will trigger an evacuation pause which is called young pause, and it is the first pause. After that, the rest live objects will be evacuated and compressed into survivor region, whose space size is always %10 of whole heap size. This mode will continue and the new objects will be allocated into eden, the young garbage collection will be triggered when the eden is filled again, and objects which achieve the specific age will level up to old generation.

3.2.3 System activity report

WANdisco system use Sar(System Activity Report) to report on various system loads, and store the report in sysinfo part of logs, they are actually periodic data of operation system.

Sar is a Unix system monitor command, it can get samples of present system status, it save system stat from multi-type device and get the review of historical performance of server. The most useful information it offers including CPU activity, memory/paging and swap space utilization. By using them, we can get the crucial figures such as %commit and %iowait, they are important for finding failure by comparing to other figures.

Chapter 4

Methodology

4.1 Programs used

python 3.6.7 with the pandas version 0.24.2 to clean and analyze the data. Matplotlib and bokeh are used to visualize the data.

4.2 Datasets

The provided data involves 2 datasets, with each dataset containing an example of a fusion server that experienced an Out Of Memory (OOM) failure. Dataset 1 originates from WANdisco's client data whereas Dataset 2 is generated via a test simulation by WANdisco. The contents of each dataset are described in the following Subsections.

4.2.1 Dataset 1

Dataset 1 contains data from a set of 4 fusion servers with the OOM failure being experienced by a node in the source zone: dc3pl3332 (see Figure 3.1). 2 nodes originated from a global production zone (source zone) and the other 2 nodes were situated in the disaster recovery zone (destination zone). The timespans for which the data is recorded for each node is given below in Table 4.1.

Node Id	Fusion server logs	GC logs	SAR stats
dc3pl3402	15 days and 7 hours	17 days and 11 hours	28 days and 22 hours
dc3pl33332	6 days and 7 hours	17 days and 11 hours	28 days and 22 hours
gdcpl4172	15 days and 3 hours	17 days and 7 hours	28 days and 22 hours
gdcpl4173	8 days and 0 hours	17 days and 11 hours	28 days and 22 hours

Table 4.1: *Dataset 1 timespan*

4.2.2 Dataset 2

Dataset 2 is similar to dataset 1 in architecture (2 nodes in each zone). The node experiencing failure is vm11. The data specifications for each node are given below in Table 4.2.

Node Id	Fusion server logs	GC logs	SAR stats
vm10	0 days and 0 hours 50 minutes	7 days and 15 hours	13 days and 17 hours
vm11	0 days and 1 hour 32 minutes	1 day and 4 hours	13 days and 17 hours
vm2	0 days and 0 hours 44 minutes	9 days and 1 hour	13 days and 17 hours
vm3	0 days and 1 hours 8 minutes	8 days and 1 hour	13 days and 17 hours

Table 4.2: *Dataset 2 timespan*

The two datasets differ in the timespans available for the pattern of OOM failure to be determined. Notably, Dataset 2's Fusion Server logs are not aligned with the GC logs and SAR stats, so the complete behaviour of the failing node, vm11 cannot be determined using Dataset 2. Furthermore, the OOM failure observed for vm11 occurs within the timeframe of 2 hour, whereas for Dataset 1 the time span is several weeks.

Since Dataset 1 originates from observed client data, and the timeframe in which failure occurs is much larger, the patterns for OOM failure are more likely to be determined. In the following sections, Dataset 1 is used to build the OOM failure prediction model and Dataset 2 is used to evaluate the model.

4.3 Data preprocessing

4.3.1 Fusion Server logs

Logs are provided in several files separated by timestamp. Each class of information are seperated by different delimiters (e.g. whitespace, :, etc). Regular expression is a sequence of characters that define a search pattern, which is suitable to extract each class in the log and convert it to structured data, which is shown in table 4.3.

timestamp	level	class	user	message
2018-11-01 15:51:31.003	INFO	com.wandisco.fs.server	teacher-3	Monitoring Manager is initialised.

Table 4.3: *Structured fusion server log*

- timestamp - the recorded time when writing the record.
- level - the log level of this message, which will be one of below.
- class - the Java class to deal with the task
- user - the thread to handle the task
- message - the description of the task, which includes the state machine, node id, dsm id, request id, request type and gsn.

Further information can be extracted from the message column and this is shown in table 4.4.

state	request id	node id	dsm id	request type
Observing	1	2	3	ConsistencyCheckDataAvailable

Table 4.4: *Further information from fusion log*

4.4 Feature Engineering

4.4.1 Feature 1: log level error ratio

As introduced in Section 4.3.1 the log level can indicate the current status of the system. The status of the node will be abnormal if the number of errors or warning messages is very high. Rather than the cumulative number of errors, the error ratio r which is the number of error messages m in a number of messages N can be used to measure the proportion of log messages corresponding to errors. This ratio can better highlight the state of the system since the number of errors will be relative to the number of processes occuring on a node. Figure 4.1 below shows a high error ratio r_{err} (0.16%) between the third of November and the fourth of November. This is approximately 10 times higher than the highest ratio in other nodes.

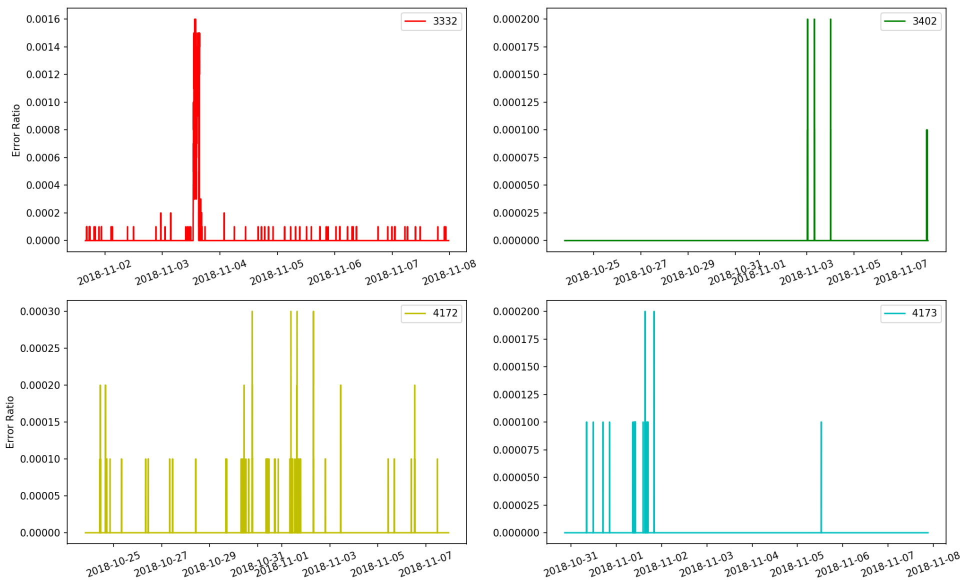


Figure 4.1: *Error ratio of four nodes*

4.4.2 Feature 2: Log level warning ratio

Similar to log level error ratio. The warning ratio r_{warn} can also highlight the state of the system. Here the disntinction between the failed node, dc3pl3332 is more apparent. From Figure 4.2, the warning ratio in the failed node is much more unstable than other nodes. The higest value reaches

100%.

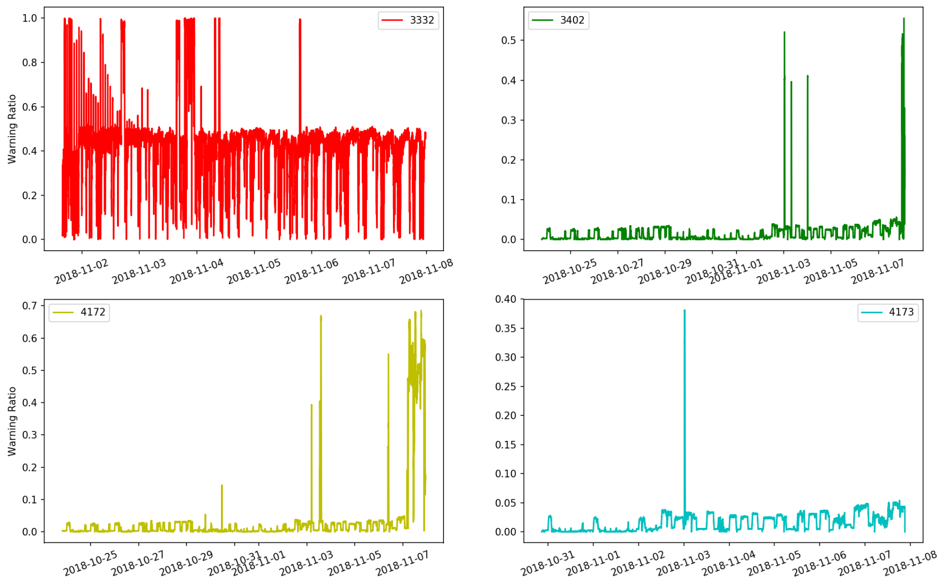


Figure 4.2: *Warning ratio of four nodes*

4.4.3 Feature 3: Commit percentage (commit %)

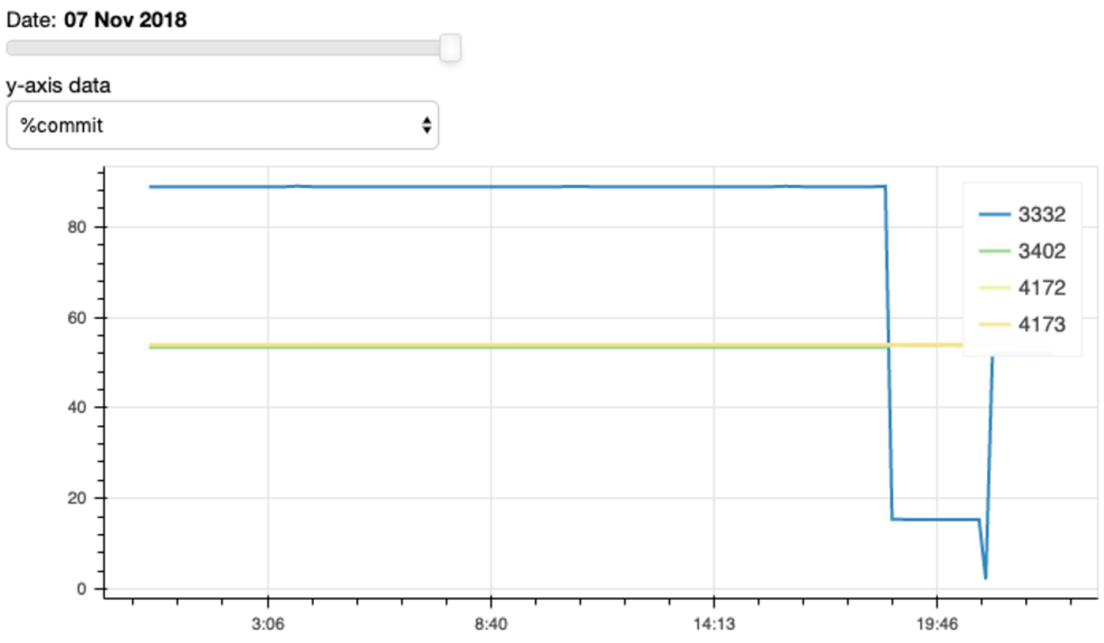
Table 4.5 is an example of system activity report:

DateDiff	node	timestamp	%memused	%commit	Data
23	3332	2018/11/1 0:10	98.97	72.66	2018/11/1

Table 4.5: *System activity report*

%commit means the percentage of memory needed for current workload in relation to the total amount of memory. It's the memory the kernel guaranteed system to use to make sure the system functions correctly. This feature in our real data are mostly stable. In our real data, most time it keeps constant except four sudden drop and jump at 21st , 23rd October and 5th , 7th November respectively, figure 4.3 is an example.

In addition, the failure node's %commit value is mostly the highest. The image shows that at 7th, the node 3332's %commit percent keeps about 30% higher than other three nodes until a sudden drop at about 18:00. Then the node is shutted down and restarted.

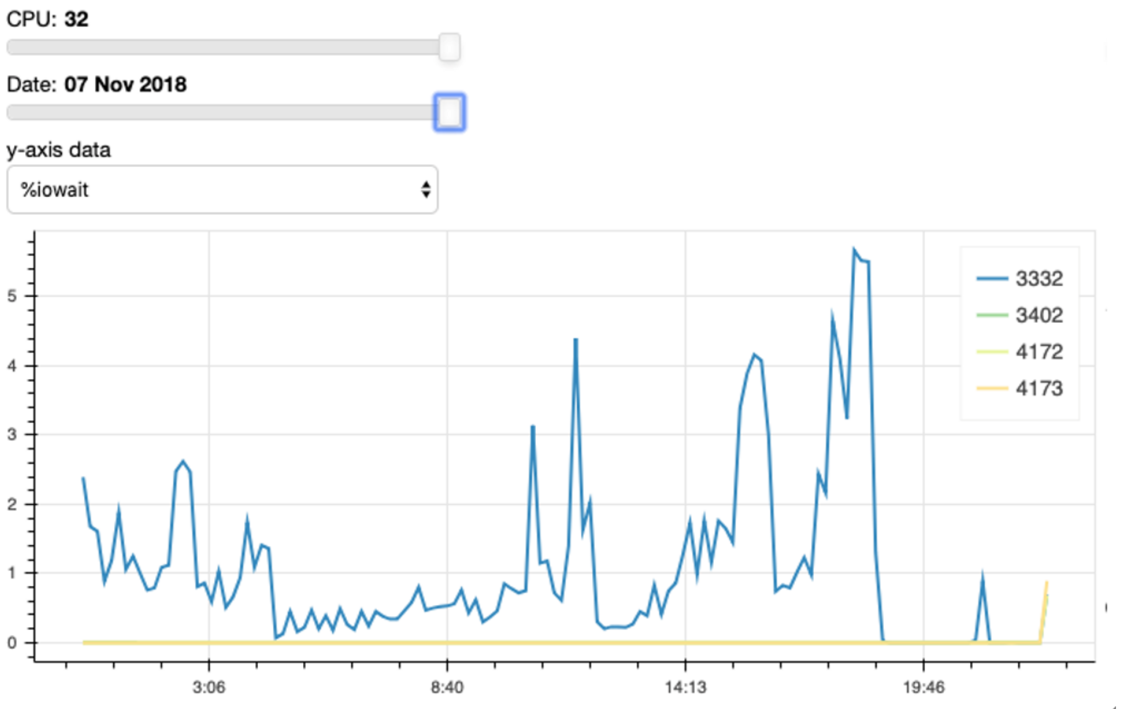

Date: **07 Nov 2018**

y-axis data

%commit

Figure 4.3: *Commit %*

4.4.4 Feature 4: IO wait


CPU: **32**

Date: **07 Nov 2018**

y-axis data

%iowait

Figure 4.4: *I/O ratio*

%Iowait is the percentage of time the CPUs were idle during a system with an outstanding disk I/O request. Therefore, %iowait means that from a CPU perspective there are no job is executable , but at least one I/O is in progress. Nothing can be arranged during iowait time. This value implies that the system is idle and is able to do more work.

Multiple reason can cause the high iowait. For example, if files are written frequently or the lack of physical memory. In our real data, Mostly, iowait for all nodes are close to 0, the failure node's iowait start increasing and fluctuating since 5th November, the image shows that it reached a peak at 7th then suddenly dropped to zero because of the failure, as figure 4.4 shows.

4.4.5 Feature 5: Garbage collection heap ratio

There is an example of gc.logs, it contains timestamp, garbage collection pause time and heap sizes change, they can figure out the process of garbage collection.

The garbage collection log contains too much messages, all information are extracted from gc.logs which contains many typical logs of evacuation pause(G1 collection), and the most important message from an individual evacuation pause is:

[Eden: 352.0M(2432.0M) $\rightarrow$ 0.0B(2400.0M) Survivors: 0.0B $\rightarrow$ 32.0M Heap: 352.0M(48.0G) $\rightarrow$ 30.3M(48.0G)]

The ratio can be computed by adding smoothing constant, $\log(x - 1 + \epsilon)$, where ϵ could be 10^{-5}.

Table 4.6 is an example of garbage collection log, it is extracted from gc.log*:

timestamp	**Heap before gc**	**Heap after gc**	**ratio**
2018/10/2 3:49:55	199MB	178MB	-2.46

Table 4.6: *garbage collection log*

It gives details about the heap sizes changes with the evacuation pause, this shows that eden had the occupancy of 352M and its capacity was 2432M before the collection, after the collection, its occupancy got reduced to 0 because everything is evacuated from eden during the collection, and its target size decreased to 2400M. The new eden capacity of 2400M is not reserved at this point. This value is the target size of the eden. Similarly, survivors had the occupancy of 0 bytes and it grew to 32M after the collection, the total heap occupancy and capacity was 352M and 48G respectively before the collection and it became 30.3M and 48G after the collection.

4.4.6 Feature 6: History size

As detailed in Section 3.1.2, state machines corresponding to replicated path requests are assigned a unique Global State Number (GSN) upon reaching a Paxos based agreement. For a given Fusion Sever, the greater the number of proposals for a replicated path (DSM), the greater the GSN range (since GSNs for proposals are increased in units). A Fusion Server continually participates in proposing, observing and executing requests. So while it can easily participate in the Paxos based agreement of proposals, it can be questioned whether the number of completed proposals is significantly lower than the number of proposals that have been agreed. A given node experiencing symptoms of OOM may be significantly behind in the proposals agreed: proposals completed scheme. Well functioning nodes may have a closer relationship between the number of proposals agreed and proposals completed. The difference between the two quantities is termed the *History size* of a given Fusion server.

For a given time interval, λ, the history size can be approximated by:

1. locating a proposal that is being observed by a fusion Server and noting the corresponding GSN.
2. locating a proposal that is being executed or has been completed within λ and noting the GSN.
3. Subtracting the two GSN.

Table 4.7 below shows two records that would be used to approximate the history size within a time interval λ (here λ can be assumed to be 30 seconds). The corresponding GSN difference is 93913, indicating that the given node is behind by 93913 proposals.

Timestamp	**State**	**GSN**	**Node ID**
2018-11-04 10:08:01,589	Observing	14776284	fcc1e83f-9aac-47de-8b79-77a7598f4c73
2018-11-04 10:08:29,225	Completed	14682371	fcc1e83f-9aac-47de-8b79-77a7598f4c73

Table 4.7: *Approximating the history size*

By varying the hyperparameter λ, the accuracy of the approximation of the history size varies. The smaller the time interval, the more accurate the approximation, however an approximation is less likely to be found. The converse is true for a larger time interval. Below, history size calculations are given for $\lambda = 10$ for all the nodes in the training data. It was observed that changing the time interval by orders of 10 only smoothes the graph but does not change the structure.

The computed GSN difference varies significantly between the failure node (dc3pl3332) and the other three nodes. The history size for the failed node is periodically exceeding 100000. dc3pl3402 also has GSN difference calculations close to 100000, but too infrequently. The other two nodes (gdc*) that are in a separate zone have an observed maximum value of 300 for the GSN difference. The history size approximation for the failure node appears to be a strong indicator for the OOM failure. Prior to the observed failure data 08/11/2018, there is a distinction in behaviour between healthy nodes and failure nodes. It should be noted however that the failed node only has log data from 01/11/2018, so the behavior prior to the increased GSN difference cannot be identified, figure 4.5 shows the details below.

4.5 Model design

4.5.1 Unsupervised learning model

Joining Features

Using the timestamp as the key, log features, garbage collection features and kernel features can be joined as a multidimensional time series dataset.

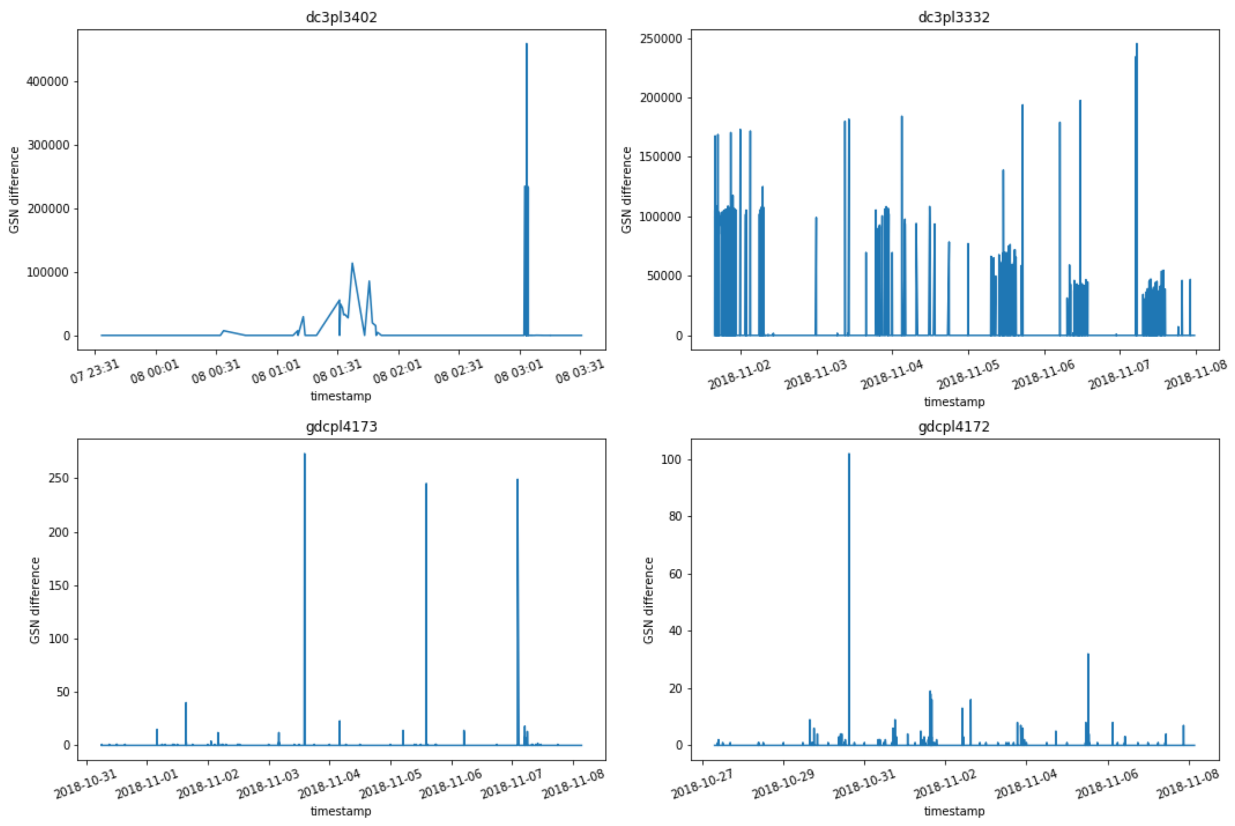


Figure 4.5: *Approximating the history size for the training data. Each nodes graph can be identified with its node ID as the title. Note that the timespan for which the GSN difference difference is computable is different for each node. dc3pl3402 only has an approximation from 7/11/2018 to 08/11/2018.*

1. For log based features including GSN difference and warning ratio, maximum values are selected within the time gap proposed in Section **??**to align the data with the CPU data.

2. For garbage collection features, the average over the past time gap is used.

Sickness Index Construction

Date	timeStamp	node	%iowait	gsn_diff	log_ratio	%commit
2018/11/7	3:10:00	3332	0.59	0	-2.944215	88.84
2018/11/7	3:10:00	4172	0	0	1.096234	53.97
2018/11/7	3:20:00	4172	0	0	1.098218	53.97
2018/11/7	3:20:00	3332	1.03	0	-2.929965	88.85
2018/11/7	3:30:00	3332	0.51	21	-2.914331	88.82

Table 4.8: *a part of structurized data*

Dataset 1 as introduced in Section 4.2.1 is used to apply the algorithms that follow. After the above merging of features for the training dataset, there are 2268 records and 7 features: log_ratio, gsn_diff, warn_ratio, %commit, %iowait, Heap_before_gc, Heap_after_gc, a subset of the features and data are shown as below, in table 4.8.

Index is a summary which can describe the overall health condition of fusion system in a numeric way. Basically, index can be thought as a feature which is derived from other features while contains as much information as possible. PCA (Priciple Components Analysis) is suitable to reduce dimension by orthogonally transforming features into linearly uncorrelated components. The idea is to use PCA to extract components firstly, then use the ratio of explained variance as weights to calculate an index for each data point.

The number of components is set to 2 for explainability. The resulting 2 principle components can be plotted to identify a difference between the transformed data points for the failed nodes data points and the healthy nodes data. The most important two components are able to explain 59.97% and 15.32% of the variance in the training data. This means the number of features are reduced from 7 to 2 with a loss of only less than 25% information. The index can be calculated by computing a weighted vector addition as shown in Section **??**.

Most features are negative to the overall health, higher GSN difference implies larger history, higher %iowait means more time are spent in idling and physical disks are saturated, higher heap size after garbage collection might be caused by diseased garbage collection mechanism. As a result, the index can be treated as a metric of sickness, the lower it is, the healthier the system is.

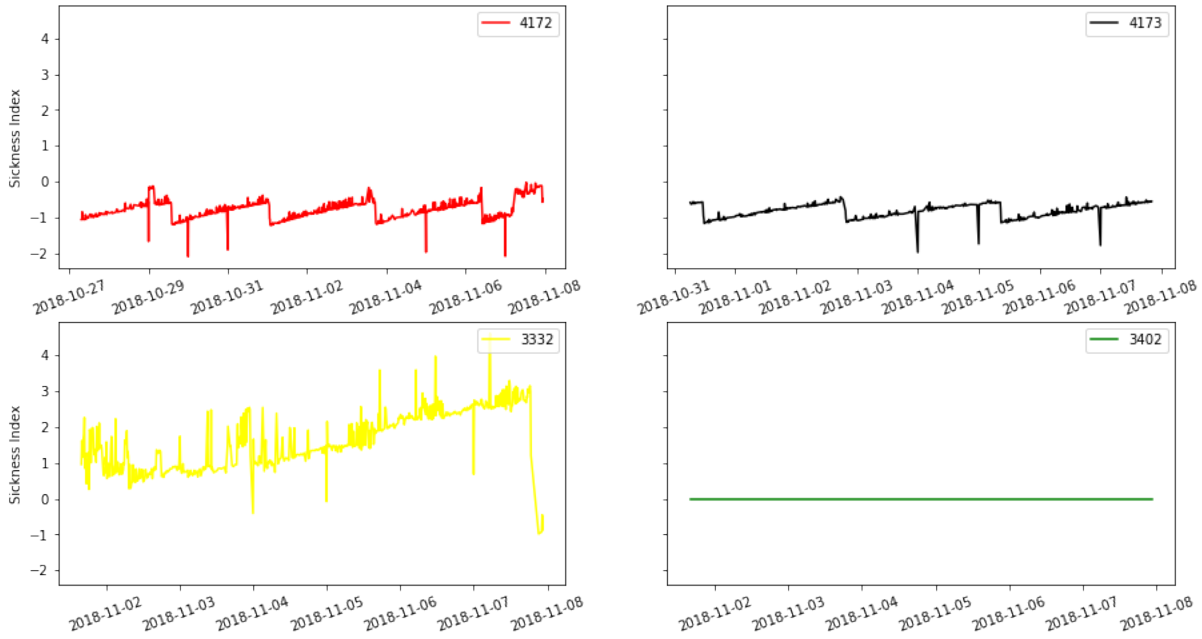


Figure 4.6: *4 nodes' sickness indices comparison*

The multi-line chart(figure 4.6) shows that both 4173 and 4172 nodes are most of the time lower than 0 while the failure node dc3pl3332 starts increasing since November the 5th and eventually dropped down suddenly because this node is killed. Node 3402's index cannot be calculated because there's no GSN difference found for this node in the period.

Time Series Segmentation

Time series segmentation is a method which can split a time series into subseries within which all data points share similar attributes. Here the time series can be segmented into "phases" which contains points close to each other so that data points can be labelled as "sick" or "healthy". This can be accomplished with Kmeans.

As a clustering algorithm, Kmeans generates random cluster centres and then minimises the inertia

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\|x_i - \mu_j\|^2 \right)$$

which stands for how data points within a cluster are close to each other by assigning the mean of each cluster as the new cluster centre until convergence.

Two components PC1 and PC2 which are extracted by PCA from the data are used as x and y axis to plot all data points. It is obvious that there are four clusters in the figure, so set the number of cluster centres as 4, the cluster results is shown at Figure 4.7.

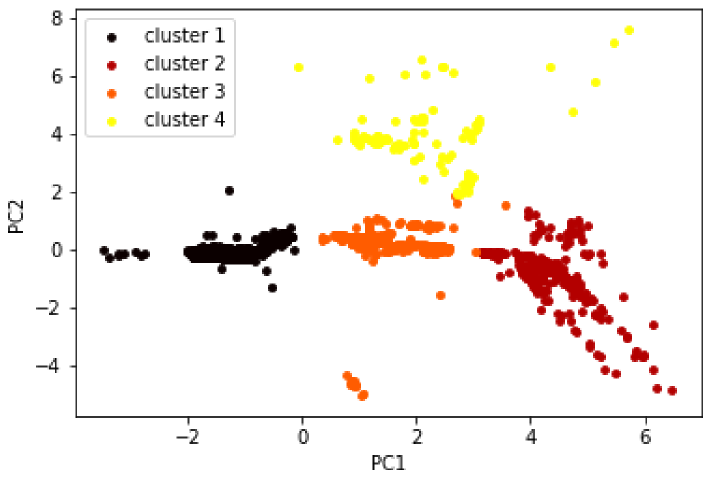


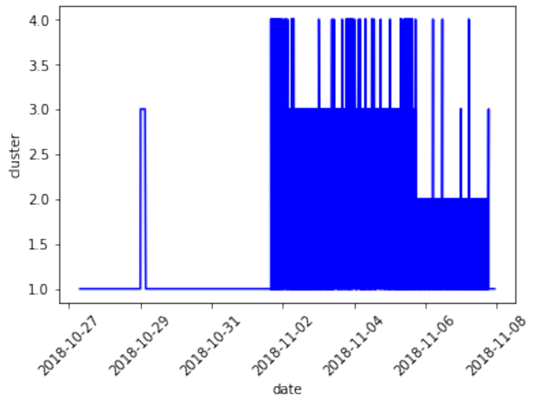
Figure 4.7: *4 nodes' sickness indices comparison*

PC	Heap_before	Heap_after	log_ratio	%commit	%iowait	gsn_diff	warn_ratio
PC1	0.88	0.96	-0.85	0.93	0.47	0.29	0.78
PC2	-0.17	-0.11	-0.06	0.01	0.55	0.80	0.29

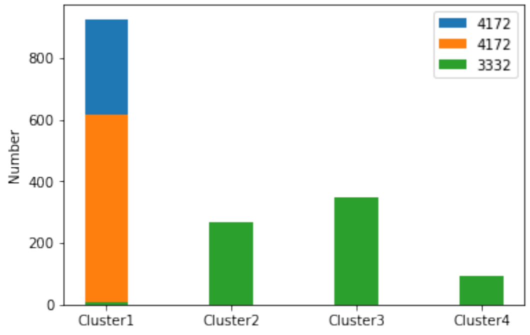
Table 4.9: *PC's correlation*

PC1 is strongly correlated with Heap_before,Heap_after,log_ratio,%commit and warn_ratio while PC2 correlated with gsn_diff, both two PCs are correlated to %iowait, table 4.9 explains the details.

Combine with Figure 4.7, cluster2 shows a high value in kernel infomation, warning rate and heap size but low in log ratio, which means the ratio of heap after garbage collection to heap before garbage collection is close to 1, which means the garbage collection system is unable to release more space.



(a) *time distribution of clusters*



(b) *node counts within each cluster*

Figure 4.8: *A comparison between clusters*

Two plots in Figure 4.8 interprets each cluster. (a) implies that data points in cluster 2 ranges from 11-05 to 11-07, with the two days before the failure node 3332 being shut down. (b) shows that cluster 2,3,4 contains only the failure node's points, that's because 3332 is the writer node, behave differently from non-writer nodes. The cluster 2 can be interpreted as "a zone in which data points' are far away from those functions correctly".

4.5.2 Supervised learning model

4.5.3 Data Labelling

In last section we've used unsupervised learning on real data set. The insights we've drawn inspired us to label cluster 2 as "1"s, which stands for "there will be a failure within next two days". Then this question can be treated as a binary classification problem. If continuous points in one node are predicted as "1"s, it's probably that such node is at a dangerous position and there will be a failure in next 2 days. This labelling method enabled us to build supervised classifiers and evaluate their performances on the testing data set.

In the real data, we used kmeans to label last 37.6% data points of the failure node as 1 s. The overall time span is 12 days while only 7 days' log for the failure node is recorded. For the testing data, we label the failure node's part in last $37.6\% * 7/11 = 22\%$ data points as failure, 1 s in the testing data means "there will be a failure in the next 10 minutes" because of the short time span.

4.5.4 Feature description

Because of the absence of garbage collection log file for the failure node, only kernel features and log-level features are available. Besides, two pair of nodes works at different time span, node vm11's failure is prior to node vm2 and vm3's log start. As it is mentioned in data set description, the log files time span lasts for less than 2 hours. We scaled testing data by sampling features with an 1 second's gap rather than 10 minutes to ensure the size of both data sets are similar.

Four features are shared by both data sets, gsn_diff, warn_ratio, %commit and %iowait.

4.5.5 Algorithm selection

As a binary classification problem, we trained logistic regression, linear SVM and RBF SVM on the cluster-labeled training set and then tested the model on the ratio-based labeled simulation testing data to see whether the model generalizes well. In all our 3 models, we set the class weight as 0.3:0.7 to assign 1 s with a higher weight because the labels is skewed on training data.

4.6 Model evaluation metrics

The simplest method to evaluate the performance of a classification model is the accuracy: the proportion of correctly classified data points. This simplicity sometimes produces unrealiable performance metrics when there is an imbalance in the labels assigned to a dataset. For example, in problems such as fraud detection, greater than 90% of the observed data is not fraudulent. A fraud classification model that achieves an accuracy of 90% is no better than a model that classifies all data points as not being fraudulent. In cases such as this, more informative evaluation metrics are required.

POWERS (n.d.) proposed taht Precision and recall are two evaluation metrics that can be used instead of accuracy to provide more informative evaluation results. Recall in the context of classification measures sensitivity. Given a data point that has a true positive label, how likely is a classifier to detect it. Precision on the other hand measures specificity, of the positively predicted data points, what is the proportion of correctly predicted (positive) data points. With these metrics, given the above fraud detection problem, the model classifying all data points as not fraudelent would receive both a precision and recall score of zero. The computation of precision (P) and recall (R) are given below:

$$P = \frac{TP}{TP + FP},\ R = \frac{TP}{TP + FN}$$

where TP = true positive, FP = false positive, TN = true negative and FN = false negative. Precision and recall can be combined to obtain a joint measure called the F_1 measure defined below:

$$F_1 = \frac{2 \times P \times R}{P + R}$$

.

In reporting the results obtained on the data to be used for evaluation, the accuracy, precision, recall and F_1 measure are reported.

Chapter 5

Results

5.0.1 Result

To evaluate the performance, we use four classification metrics: accuracy, precision, recall and f1 score. The result is shown as follows, in table 5.1:

metric	Logistic Regression	Linear SVM	RBF SVM
Accuracy	0.85	0.85	0.86
Precision	0.00	0.00	1.0
Recall	0.00	0.00	0.01
F1 score	0.00	0.00	0.01

Table 5.1: *Evaluation results for 4 features' model*

This model can reach a high accuracy but all other metrics are 0s. That's because it simply predicts each moments' label as 0. Coefficients is the weight vector which will be multiplied with features in logistic regression's equation and their values are learned during optimization. The positive higher coefficient implies that this feature is relatively important to "1" label and vice versa. The coefficients for all features are shown in table 5.2.

Because the testing data set only lasts for less than two hours within which the %commit remains constant for both nodes, the classifier is not able to identify data points which are labelled as 1 s. In order to utilize other features, we removed %commit to build other models.

features	%iowait	%commit	gsn_diff	warn_ratio
coefficients	0.60	5.09	-0.76	-1.46

Table 5.2: *Evaluation results for 4 features' model*

As the feature with the highest weight, a high %commit value can be differed from data points labeled as normal. The %commit value for testing data is plotted in figure 5.1:

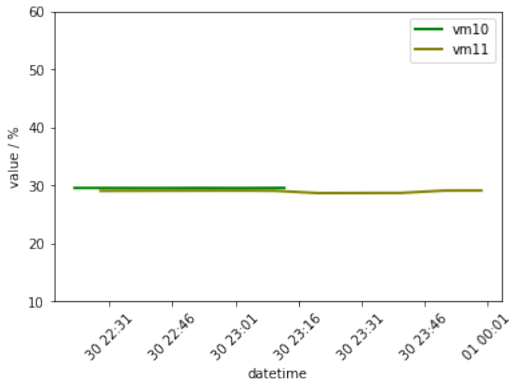


Figure 5.1: *4 nodes' sickness indices comparison*

Then we compared the 3-feature classification models, SVMs performances are significantly higher than logistic regression in table 5.3. RBF kernel SVM can perform slightly better than linear kernel.

metric	logistic regression	linear SVM	RBF SVM
Accuracy	0.92	0.95	0.95
Precision	0.93	1.0	1.0
Recall	0.51	0.64	0.67
F1 score	0.66	0.78	0.80

Table 5.3: *Evaluation results for 3 features' models*

After we removed the %commit feature, models can effective classify data points on more time-sensitive features like warn_ratio and %iowait. the coefficients for three features are shown in table 5.4, weights are more balanced and the %iowait became the most important feature.

features	%iowait	gsn_diff	warn_ratio
coefficients	1.91	-0.38	1.33

Table 5.4: *Evaluation results for 3 features' logistic regression*

Chapter 6

Discussion

The proposed models are able to distinguish the Fusion Server experiencing an OOM in the training dataset and test dataset. Though the positive label of “failure within 2 days” is scaled to obtain a meaningful representation in the test data, “failure within 10 minutes”, all trained models are able to achieve F_1 scores of atleast 66%. The ability of the models to distinguish nodes experiencing symptoms of an OOM failure and healthy nodes is evident based on the features engineered. Each feature highlights a key difference in behaviour between a node experiencing OOM failure and a healthy node.

The top features that are able to distinguish between nodes experiencing OOM and healthy nodes can be realised by using PCA. In Section **??**] the top principal component, PC1 which explains 60% of the variation in the data is strongly correlated with Heap before GC and %commit. This could because of the relationship between garbage collection and committed memory. High heap size after garbage collection indicates that a large proportion of commited memory is occupied by the JVM garbage.

The PCA based “sickness” index further supports the above. The data between nodes experiencing OOM and healthy nodes is sufficiently varied, and this allows the index of the failing node to be greater than the other nodes (see Section **??**). The PCA based feature is therefore a valuable indicator for a node experience OOM, and this can satisfy the role of summarising the health of Fusion servers when a quick indication is needed.

Though the supervised prediction models proposed have a high performance on the test dataset. the generalisability of the developed models can be questioned due to the inherent difference in the datasets used to train and evaluate the models. As documented in Section **??**, Dataset 2 only has data available for at most 1 hour whereas dataset 2 has data spanning a week. The type of OOM failure experienced in Dataset 2 may be of a more instant nature in comparison to the failure observed in Dataset 1. The conditions in which the data are constructed is also different, and this could be the reason features such as the commit % are not as informative in distinguishing between the healthy and failure nodes in dataset 2.

Chapter 7

Conclusions

A data preprocessing scheme is provided to generate structured data from the fuision server logs that can be used to extract statistics and information about the health of Fusion Servers in relation to whether they are experiencing symptoms of OOM.

Models are proposed: require futher analysis on datasets with similar distributions of failure to test generalisability. The model which utilised 3 features can generalize well on the internal testing data. However, more real world data is needed to evaluate the true generalizability of our models.

The PCA based index will be useful to indicate the health situation's tendency of nodes. This index can be implemented with online system like dashboards to actively monitor the state of fusion servers with respect to OOM failures.

Lastly, features are identified that %commit and heap_after_gc are strongly correlated with the OOM symptoms on a failing node. Also, all the features we extracted show collinearity, all of them are related to the fusion servers' sickness.

Bibliography

Abeyasekera, S. (2003), 'Multivariate methods for index construction', *Household surveys in developing and transition countries: design, implementation and analysis* .

Adewale, A. J., Dinu, I. & Yasui, Y. (2010), 'Boosting for correlated binary classification', *Journal of computational and graphical statistics* **19**(1), 140–153.

Bewick, V., Cheek, L. & Ball, J. (2005), 'Statistics review 14: Logistic regression', *Critical care* **9**(1), 112.

Cocagne, T. (2016), 'Understanding paxos introduction'.

Detlefs, D., Flood, C., Heller, S. & Printezis, T. (2004), Garbage-first garbage collection, *in* 'Proceedings of the 4th international symposium on Memory management', ACM, pp. 37–48.

Donaubauer, J., Meyer, B. E. & Nunnenkamp, P. (2016), 'A new global index of infrastructure: Construction, rankings and applications', *The World Economy* **39**(2), 236–259.

Fronza, I., Sillitti, A., Succi, G., Terho, M. & Vlasenko, J. (2013), 'Failure prediction based on log files using random indexing and support vector machines', *Journal of Systems and Software* **86**(1), 2–11.

Fulp, E. W., Fink, G. A. & Haack, J. N. (2008), 'Predicting computer system failures using support vector machines.', *WASL* **8**, 5–5.

Gurumdimma, N., Jhumka, A., Liakata, M., Chuah, E. & Browne, J. (2015), Towards detecting patterns in failure logs of large-scale distributed systems, *in* '2015 IEEE International Parallel and Distributed Processing Symposium Workshop', IEEE, pp. 1052–1061.

Hautamaki, V., Nykanen, P. & Franti, P. (2008), Time-series clustering by approximate prototypes, *in* '2008 19th International Conference on Pattern Recognition', IEEE, pp. 1–4.

Hosmer Jr, D. W., Lemeshow, S. & Sturdivant, R. X. (2013), *Applied logistic regression*, Vol. 398, John Wiley & Sons.

Huang, X., Ye, Y., Xiong, L., Lau, R. Y., Jiang, N. & Wang, S. (2016), 'Time series k-means: A new k-means type smooth subspace clustering for time series data', *Information Sciences* **367**, 1–13.

Jain, A. K. (2010), 'Data clustering: 50 years beyond k-means', *Pattern recognition letters* **31**(8), 651–666.

Kejariwal, A. (2013), A tool for practical garbage collection analysis in the cloud, *in* '2013 IEEE International Conference on Cloud Engineering (IC2E)', IEEE, pp. 46–53.

Komarek, P. (2004), 'Logistic regression for data mining and high-dimensional classification'.

Lavinia, A., Dobre, C., Pop, F. & Cristea, V. (2011), 'A failure detection system for large scale distributed systems', *International Journal of Distributed Systems and Technologies (IJDST)* **2**(3), 64–87.

Liang, Y., Zhang, Y., Xiong, H. & Sahoo, R. (2007), Failure prediction in ibm bluegene/l event logs, *in* 'Seventh IEEE International Conference on Data Mining (ICDM 2007)', IEEE, pp. 583–588.

Mayr, A., Binder, H., Gefeller, O. & Schmid, M. (2014), 'The evolution of boosting algorithms', *Methods of information in medicine* **53**(06), 419–427.

Montgomery, D. C., Jennings, C. L. & Kulahci, M. (2015), *Introduction to time series analysis and forecasting*, John Wiley & Sons.

POWERS, D. M. W. (n.d.), 'Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation', *Journal of Machine Learning Technologies* **2**.

Reiss, S. P. (2009), Visualizing the java heap to detect memory problems, *in* '2009 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis', IEEE, pp. 73–80.

Rodriguez, A. C. & de los Mozos, M. R. (2010), Improving network security through traffic log anomaly detection using time series analysis, *in* 'Computational Intelligence in Security for Information Systems 2010', Springer, pp. 125–133.

Samak, T., Gunter, D., Goode, M., Deelman, E., Juve, G., Silva, F. & Vahi, K. (2012), Failure analysis of distributed scientific workflows executing in the cloud, *in* 'Proceedings of the 8th international conference on network and service management', International Federation for Information Processing, pp. 46–54.

Schapire, R. E. (2003), The boosting approach to machine learning: An overview, *in* 'Nonlinear estimation and classification', Springer, pp. 149–171.

Schneider, F. B. (1990), 'Implementing fault-tolerant services using the state machine approach: A tutorial', *ACM Computing Surveys (CSUR)* **22**(4), 299–319.

Suykens, J. A. & Vandewalle, J. (1999), 'Least squares support vector machine classifiers', *Neural processing letters* **9**(3), 293–300.

Tipping, M. E. & Bishop, C. M. (1999), 'Probabilistic principal component analysis', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **61**(3), 611–622.

Vlachos, M., Lin, J., Keogh, E. & Gunopulos, D. (2003), A wavelet-based anytime algorithm for k-means clustering of time series, *in* 'In proc. workshop on clustering high dimensionality data and its applications', Citeseer.

WANdisco (2019), 'Wandisco fusion talkbacks'.

Zheng, Z., Lan, Z., Park, B. H. & Geist, A. (2009), System log pre-processing to improve failure prediction, *in* '2009 IEEE/IFIP International Conference on Dependable Systems & Networks', IEEE, pp. 572–577.