Mobile Publish/Subscribe System for Intelligent Transport Systems over a Cloud Environment

Robayet Nasim, Andreas J. Kassler
Department of Mathematics & Computer Science
Karlstad University
SE-651 88 Karlstad, Sweden
robayet.nasim@kau.se, andreas.kassler@kau.se

Ivana Podnar Žarko, Aleksandar Antonic
Faculty of Electrical Engineering and Computing
University of Zagreb
HR-10000 Zagreb, Croatia
ivana.podnar@fer.hr, aleksandar.antonic@fer.hr

Abstract— **With the advent of Smart Cities, public transport authorities are more and more interested in Intelligent Transport System (ITS) applications that allow to process a large amount of static and real time data in order to make public transport smarter. However, deploying such applications in a large scale distributed environment is challenging and requires an automated, scalable, flexible, elastic, loosely-coupled communication models in order to dynamically link information providers and consumers. To this end, Publish/Subscribe (Pub/Sub) systems offer an asynchronous, dynamic, decoupled interaction scheme that is perfectly suitable for developing up-to-date, large-scale distributed applications within the ITS domain. In addition, cloud computing offers computational resources as services to utility driven model regardless of considering geographical locations in a scalable, elastic, fault tolerant and cost-effective way. In this work, we build an ITS application "Real-time Public Transit Tracking" on top of a Mobile Pub/Sub System (MoPS), and deploy it over an open source cloud platform, OpenStack, in order to achieve high performance and flexible management. We conduct a set of experiments to evaluate the performance of the implemented ITS application in terms of scalability, resource usage, and efficiency of the underlying matching algorithm under automated mobility of the subscribers. Our experimental results show that the ITS application can handle a large number of subscribers and publishers with proper reliability and negligible notification delay under real-time constraints. Further, we present a measurement study to characterize the impact of different workloads on the performance of OpenStack.**

Keywords— ***Publish/Subscribe System; Mobility; Cloud Computing; OpenStack; Intelligent Transport System (ITS).***

I. INTRODUCTION

With the rapid increase of people's dependence on public transport systems in large cities extends the probability of deploying ITS to a large extent. ITS already acts as an efficient way of improving public transportation system as it reduces traffic congestion and enhances travel flexibility. But in order to make ITS operate efficiently, the architecture needs to be designed effectively, specially designing services for end users under real-time and mobility constraints.

A Pub/Sub system is an asynchronous communication mechanism among a set of data producers and consumers. Producers publish information and consumers show their interest in certain types of information in terms of subscriptions. The Pub/Sub middleware is responsible for filtering, matching and delivering the information to the interested subscribers. The increasing demands for efficient, reliable, flexible communication patterns for large scale distributed systems make the concepts of Pub/Sub systems interesting to be applied as building blocks of ITS applications. Pub/Sub systems have two important characteristics that make them very interesting to be deployed on a large scale: "loose coupling" and "asynchronous push-based communication" [1]. These characteristics are very useful for ITS applications because distributed transportation departments, heterogeneous devices and different data sources within ITS domain require extensive integration for providing integrated services to the end users.

The Pub/Sub systems can be divided into two general categories [2]: subject-based and content-based. In subject-based systems, subscribers subscribe to specific topics and receive all publications published to that topic. On the other hand, the content-based Pub/Sub system [6] is more flexible and expressive language. It enables subscribers to define their interests based on the contents of publications.

In order to apply Pub/Sub systems in the area of Intelligent Transport Systems (ITS), two main issues have to be solved. ITS applications typically have to deal with the continuous publication update of large volume of real-time data, which might impose scalability issues. Second, ITS applications may require automated context aware message delivery to a large numbers of mobile subscribers. Consider for example an ITS application [4] that informs mobile users with real-time vehicle positions of busses and trains. In a large city such as New York or London there may be thousands of such vehicles periodically updating their position and status and millions of mobile users interested in such information. However, different users are interested in different kinds of information, which may change over time. Therefore, the ITS applications require to keep up with the changing demands in an automated and efficient way. Finally, the Pub/Sub system needs to be elastic and fault tolerant in order to efficiently cope with sudden changes in the workloads at certain periods of time at certain location, and handle sudden crash of subsystem. For instance, during peak hours in some central locations of a city, the number of subscribers and vehicles may increase at an exponential rate. The deployment of the system should be efficient and simple so that it can automatically adapt to the changes in the environment or load.

Cloud computing offers through which e.g. computing resources such as infrastructure, platform applications, or business processes can be provided to consumers as a service without depending on the locations of the consumers. This reduces in-house operational cost by utilizing resources properly, and by handling dynamic resource allocation efficiently through scaling the system according to load. There are a large number of open source solutions for cloud platforms and OpenStack [8, 13] is an important Cloud platform because it is easy to setup for private cloud environments, and shows high performance, scalability, elasticity and automated management of the resources. OpenStack uses a suite of software components [7, 8, 13] and is compatible with Amazon EC2[1] and Amazon S3[2].

This paper evaluates a real-time ITS application over MoPS [5], a distributed content based Pub/Sub system, within our OpenStack private Cloud testbed. MoPS consists of a set of interconnected brokers and supports mobility of subscribers by storing valid notifications for disconnected brokers. Publishers are given the opportunity to set a validity period for their publications. This paper makes three contributions. First, we build an ITS application where all the vehicles in a region publish their real-time positions and subscribers register their interests by sending subscriptions which are a logical combination of several rules based on discrete attributes. The ITS application is designed in such a way that it can handle the mobility of the subscribers automatically. Secondly, we evaluate the scalability of the system representing real-life scenarios, by varying the number of publishers, subscribers, publication rate, and mobility of subscribers where workload is generated through automated scripts. Thirdly, our study provides insights on resource sharing on virtual machines that are valuable to administrators running variety of real-time applications on top of OpenStack.

The rest of the paper is organized as follows: Section II depicts the brief architecture of our ITS application, MoPS and OpenStack testbed. Section III describes the implementation of the ITS application, explains the experimental setup and discusses evaluation results. Section IV discusses related work. Lastly, Section V concludes the paper and outlines future plans.

II. SYSTEM OVERVIEW

A. *ITS Application*

Adapting the cloud paradigm for ITS applications is still in its infancy and several parameters need to consider carefully such as - mobility of the users, underlying protocol for the Pub/Sub system, performance and designing of the matching algorithm running on the broker, resource allocation strategy of the cloud, intra cloud communication, communication between homogeneous and heterogeneous clouds, etc. In order to accommodate a real-time large scale ITS application, we propose a system by combining pub/sub messaging paradigm with cloud computing, depicted in Fig. 1. Figure 1 shows the development view of the ITS application "Real-time Public Transit Tracking. Our system is made out of three fundamental components: *Publisher/Subscriber, MoPS Broker* and *OpenStack*. One of the fundamental objectives of our system is to be scalable with underlying infrastructure. The lower part of the figure presents the physical nodes within our private cloud deployment. Multiple MoPS brokers are deployed on virtual machines (VMs) of the physical nodes. The number of brokers can be increased by deploying new VMs in order to handle new requirements on workload and scalability. Publishers and subscribers are deployed on different VMs which generate publications and subscriptions in an automated way.

B. *Pub/Sub System Model*

MoPS [5] is a broker-based distributed content based Pub/Sub system [6] where all publishers and subscribers communicate through the brokers. The broker node is the core part of the system, which matches and delivers information from the publishers to the subscribers. One publication can be a set of arbitrary attributes where the numbers and values of the attributes can vary within a broad range. Each publication supports two types of attributes – Numeric and String. Moreover, all the publications contain a timestamp to specify the validity period so that the brokers can remove the publication after the validation expires. The subscriptions can be formed by setting different queries on these attributes' values. The number of MoPS brokers depends on workloads and all the brokers may be connected to form an acyclic communication graph or can be deployed independently. If all the brokers are connected, they inform each other about their subscriptions to have a consistent view of the active subscriptions of the whole system, to avoid duplicate delivery of the notifications. To cope with scalability, the number of brokers can be increased by connecting a new broker to any of the existing brokers. The mobility of the publishers and/or subscribers is managed within MoPS through a sequence of commands – *connect, disconnect, reconnect*. TCP is used as the underlying communication protocol within MoPS. The communication between a subscriber and a broker is broker-initiated that is when a broker has publications that match with any subscriptions, the messages are automatically pushed to the subscriber.

C. *Cloud Infrastructure*

We deploy the brokers within Virtual Machines in our Cloud Infrastructure, which is based on OpenStack. OpenStack consists of five different projects [7, 8] – Compute (Nova), Image (Glance), Storage (Swift), Dashboard (Horizon), and Identity (Keystone). Nova [7, 8] manages the life cycle of a VM from allocation of computing resources,

[1] http://aws.amazon.com/ec2/
[2] http://aws.amazon.com/s3/

network management, etc. to restart or destroying the VM. It has six components to offer different services – Nova API, Message Queue, Nova-Compute, Nova-Network, and Nova-Scheduler. Nova API service takes the commands from the cloud-consumers and performs the actions in the cloud. Nova Compute service is responsible for creating and managing the VMs in a physical server. Nova Network service manages all the tasks related to network and communication among the VMs and between the physical host and a VM. Nova Scheduler service is responsible for scheduling the VMs in the physical servers. Swift [7, 8] provides distributed storage infrastructure for OpenStack. Glance [7, 8] is responsible for managing image-related tasks such as creating snapshots of running VMs. Horizon [7, 8] provides a browser-based user-friendly interface for managing and maintaining all the services provided by OpenStack. Keystone [7, 8] provides role-based authentication and authorization functions for the OpenStack users such as admin or member of the project.

III. Implementation and Evaluation

A. Implementation

We have implemented an ITS application, "Real-time Public Transit Tracking" on top of MoPS. Each vehicle is considered as a publisher, publishing its current position at regular intervals. For publication format, we have used the "Vehicle Positions" specification from General Transit Feed Specification (GTFS) [10]. GTFS-real-time is a feed specification that allows public transportation agencies to provide real-time updates about their fleet (e.g., buses or trains) to application developers. The GTFS-real-time specification is based on Google's protocol buffers[3], a lightweight data interchange format comparable to XML. Our implementation uses Java. We have implemented a class called "VehilcePositionsPublication" which contains the attributes – Latitude, Longitude, Bearing, Odometer, Speed, VehicleID, VehicleLabel, LicensePlate, VehilceStopStatus, CongestionLevel to represent one publication. The period of each publication is valid until the vehicles update their positions through submitting new publications. Subscribers define their subscriptions by putting a set of queries on publication attributes where each query consists of an attribute, value and operator. Mobility is one of the important characteristics for any ITS application. We have implemented the mobility of the subscribers by changing the subscriptions, and either by connecting them to the same broker, or by reconnecting to a new broker automatically.

B. Experimental Setup

For the cloud environment, we have installed OpenStack in Ubuntu 12.04 LTS server edition. Two-node cluster are used to setup the OpenStack private cloud environment. One node is used as the "Management node" and another node is used as "Compute node" where the first one is used for running the management console to control the VMs that are running on both machines. The "Management node" has the following configuration: Intel(R) core(TM): 7-3770 CPU@3.40 GHZ, 32 GB RAM, 250 GB HD and the "Compute node" has the configurations: Intel(R) Xeon(R) CPU E5540@2.53GHZ, 160 GB RAM, 2 TB HD. The link speed between the nodes is 100 Mbit/s. In our OpenStack testbed, Kernel-based Virtual Machine (KVM) is used as hypervisor. The Architecture of our experimental setup is presented in Fig. 2. Each broker is deployed on a VM and all of them have the same resource allocation - 8 GB memory, 4 VCPU, 40 GB storage. Moreover, we have deployed other 10 VMs (one for each broker) to generate automated impact

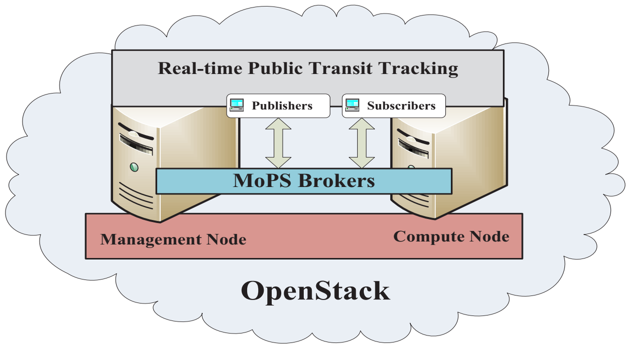

Figure 1. Development View of Real-time Transit Tracking

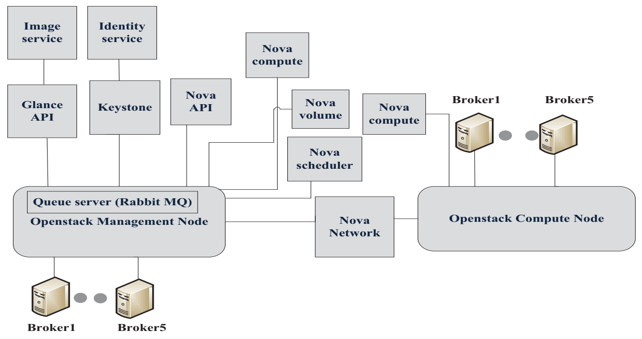

Figure 2. Architecture for Experimental Setup

TABLE I. Fixed Parameters in Bokers' Machines

Parameters	Values
net.ipv4.tcp_sack	1
net.ipv4.tcp_rmem	1024 87380 67108864
net.ipv4.tcp_wmem	1024 87380 67108864
net.ipv4.tcp_congestion_control	vegas
net.ipv4.tcp_mtu_probing	1
net.ipv4.tcp_low_latency	0
net.ipv4.tcp_window_scaling	1
net.ipv4.tcp_moderate_rcvbuf	1

on the performance of the TCP. The parameters "net.ipv4.tcp_rmem" and "net.ipv4. tcp_wmem" are used to change the minimum/default/ maximum receive and send buffer sizes for tcp. For congestion avoidance algorithm (net.ipv4.tcp_congestion_ control) we have selected to use TCP Vegas because of its emphasize on reducing packet delay. We have enabled TCP MTU probing to use progressive TCP packet sizing if there is a router which does not support default MTU size. We have used the parameter

[3] https://developers.google.com/protocol-buffers/

"net.ipv4.tcp_low_latency" to select low latency over higher throughput for TCP. The parameter "net.ipv4.tcp_ window_scaling" is used for both receive and send window to TCP window size greater than 64 KB and the parameter "net.ipv4.tcp_moderate_rcvbuf" is used for auto-tuning the receive buffer to scale the socket memory within the range specified in the parameter "net.ipv4.tcp_rmem".

We have considered two scenarios to investigate the performance of the ITS application "Real-time Public Transit Tracking" over OpenStack. We have deployed in total 10 VMs, each one running an independent Broker, five of which are deployed on the management node and five on the compute node. The Brokers are independent and not connected with each other. Each publisher is considered as an individual vehicle which publishes its current position at regular intervals to a broker and the subscribers register their random subscriptions. We have used dedicated brokers for different vehicle lines such that a vehicle running on line number 1 publishes its positions to broker 1 where as a vehicle operating on the line 10 publishes its position to broker 10. We map the mobility of the subscribers by changing the subscriptions of the travelers. We argue that as users move around the city their context changes and thus also their subscriptions. We have considered two scenarios for presenting mobility where in the first scenario, 1000 subscribers change their subscriptions in the middle of the experiments but connect to the same broker (e.g. users are interested on the same line but in different area). In the second scenario, 1000 subscribers reconnect to a new broker automatically with new subscriptions (e.g. users change line on the way of their travel).

In the experiments, every publication simulates the real-time GTFS data for one vehicle. Each publication is different from another publication in terms of values of the attributes. For instance, publisher 2 publishes random publications where values of the attributes are different from the publications of the publisher 1. In all the experiments except the last one, the same number of subscribers is connected to each broker to generate same workload on the brokers. Subscriptions are conjunction of predicates where the predicates are combinations of attributes' values and comparison operators. Again for all the experiments except the last one, we have used either 2500, or 5000, or 10000 subscribers for each broker with random subscriptions. We evaluate the system by letting the publishers issue 30 publications and we vary the publication rate from 1 publication per 60, 90 or 120 seconds. In addition, we vary the number of publishers from 1000 to 5000. The last experiment compares the performance of our system for different workloads on three brokers with the same workloads on the same brokers. This test also investigates how OpenStack shares resources among VMs for similar and dissimilar workloads whereas they are allocated the same resources during the time of launching.

C. Evaluation

We have used a number of metrics in the experiments in order to measure the performance of our system. The "End-to-end Delay" is the average latency between publishers issuing publications until all the subscribers of interest have received it. The "Processing Time" defines the duration for matching single message in the brokers. The "Memory Usage" is the total memory consumption of the brokers' VMs after the experiments for each combination of publishers and subscribers. We have used the default linux command "free" for the memory consumption in the VMs. The "Response time" is the duration for sending notifications to a new subscriber that reconnects to a new broker with new subscriptions. The "Network Load" is measured by calculating the total number of packets captured within a broker VM, and also by calculating the average packet sending rate from the brokers. We have used the tool "tcpdump" to capture the packets on the brokers' listening port and used the tool "tcpstat" to report the statistics.

Scalability: The scalability of MoPS is evaluated by considering the capacity of handling large number of publishers and subscribers when varying the publication rate. During the time of evaluation, we have focused on three key points: 1) how does the number of publishers, publication rate and number of subscribers affect the notification time? 2) how does the matching algorithm perform for different sets of subscriptions? 3) how does the performance change when increasing the matching probability of subscriptions? The workload is self-similar on every broker as there are no dependency between publications and subscriptions among different brokers. Figs. 3-5 show the notification delay for different number of subscribers for scenario 1, where Fig. 6-8 represent the notification delay for scenario 2. In both cases, we observe a small increase in the delivery time with the increase of publishers, subscribers and publication rate. It clearly states that an increasing number of publishers or subscribers raises workload on the brokers, where on the other side increasing the interval between the publications decrease the load on the brokers by providing them more time to process prior messages. The standard deviation of the notification delay depends on the hosts where they are deployed as the host nodes are different in terms of underlying hardware resources, number of VMs currently running, which impact on the resource allocation. We observe that the notification time does not spread over a long range. Figs. 9-11 show the notification delay when varying the matching probabilities, as for example a publication needs to be delivered to 30%, 50% and 100% subscribers. As expected, from these results we can see that the notification delay is increasing with increasing matching probability of the subscriptions. More importantly, we can make an interesting observation from these graphs. Large numbers of publishers and subscribers have more impact on notification delivery of the brokers compared to small numbers of publishers or subscribers, but the overall impact is still satisfactory for large scale real-time applications.

Mobility and Churn: Initially the number of subscribers connected to each broker is same. But changing subscriptions in our both scenarios represent the mobility of the subscribers. Moreover, scenario 2 introduces churn on the system by adding some subscribers on a broker at the middle of the experiments which changes the workload suddenly. Therefore, the notification delay for scenario 2 is the same as response time that we mentioned earlier. From the results, we observe that the notification delay for scenario 2 is higher than scenario 1. It is reasonable because in the latter case the subscribers send new subscriptions to a new broker when that broker is serving active subscribers. Therefore, there is some waiting delay before the broker starts processing new subscriptions. In the worst case, for 10000 subscribers and for 5000 publishers with 60 seconds publication interval, the notification delay is around 2 seconds, which is still acceptable in the context of real-time ITS applications.

Brokers' Load: In OpenStack, the resource usage on different VMs (Brokers) have the same tendency. We have measured the memory usage, total number of packets and packet rate on all the brokers during the time of the first set of experiments (when the brokers need to deliver notifications to 30% of the subscribers) to check the brokers' performance in terms of resource consumption. Results are presented in Figs. 12-14, which indicate that memory usage, total number of packets and packet rate, increases with increase in the number of publishers and subscribers. To test the performance of the real-time event matching capacity we perform a series of experiments by setting different numbers of rules for the subscriptions. For the simple case, the subscription is composed of a rule on a single attribute. For more complex subscriptions, we specify conjunctions of two rules based on two attributes. To form the most complex subscriptions, we combine four different rules based on four different attributes. Figs. 15-17 depict brokers' processing time for different number of subscribers along with different level of complexity for subscriptions. The results show very slow increase in matching time of the brokers with the matching complexity level. The matching algorithm works very efficiently against complex set of subscriptions. Fig. 18 demonstrates brokers' processing time for the most complex set of subscriptions for different number of subscribers. Observing Fig. 18, we can see that broker's processing time increase very slowly with the number of subscribers.

Load Balancing: The main factor that affects the brokers' processing time and notification delay is the amount of workload on the VMs. In this experiment, we evaluate the impact of equal and unequal workloads on VMs when the total amount of workload is the same at the physical node where the OpenStack Compute package is deployed. During the time of evaluation we have focused on the question how effectively OpenStack VMs share resources. We deploy in total 3 VMs, each one running one broker and use 2500 subscribers per broker to generate identical workload (in total 7500 subscribers). We compare this with a scenario where we create different workload per broker (and thus VM) by using 1000, 2500, and 4000 subscribers per broker (in total 7500 subscribers). While the workload per VM is different, the aggregated workload per physical host is the same in both cases. The results presented in Figs. 19-20 indicate strong similarity in notification delay and processing time for the brokers between similar and dissimilar workloads. Figs. 19-20 show that fluctuating workload on the VMs have very small impact on the performance of OpenStack. Even for the case of 5000 publishers, distinct workloads per broker gives relatively good result compared to identical workload per broker. More importantly, we can make a valuable observation that OpenStack can optimize the workloads' performance by utilizing physical resources efficiently.

IV. RELATED WORK

A big part of the current publish-subscribe system and stream processing research is till oriented towards distributed systems, but very few researchers have started to use the cloud approach as well. But using cloud platform and a pub/sub system for dealing with a massive data exchange in Smart Cities scenarios is still not focused properly. One general-purpose Pub/Sub architecture in combination with a cloud is described in Li M. et al. [3]. It offers an attribute-based Pub/Sub system, BlueDove which is based on the idea of cloud-based Pub/Sub service. It offers a two-tier architecture which combines the components, "matcher" and "dispatcher". In one tier services are exposed through Internet by the component called dispatcher. These dispatchers have public interface so that publishers and subscribers can connect with them directly and send the publications and subscriptions. In another tier the dispatchers are connected with another component called matcher where the matchers are responsible for delivering notifications to the correct subscribers. The simple on-hop look up for locating the correct matcher increases the overall throughput of the system as well as reduces the overhead. BlueDove eliminates the necessity of broker-to-publisher or broker-to-subscriber affinity. In [3], the authors evaluate the scalability, fault tolerance, elasticity of BlueDove through a large number of experiments and compare the results with two different approaches – peer-to-peer Pub/Sub system and full replication method. The result shows that BlueDove can handle more workloads compared to the other two approaches. This work is very much similar to our approach but it does not consider the mobility of the clients which is very important characteristic for any ITS application. Moreover, maintaining global view of the system among all components creates overhead for the whole system.

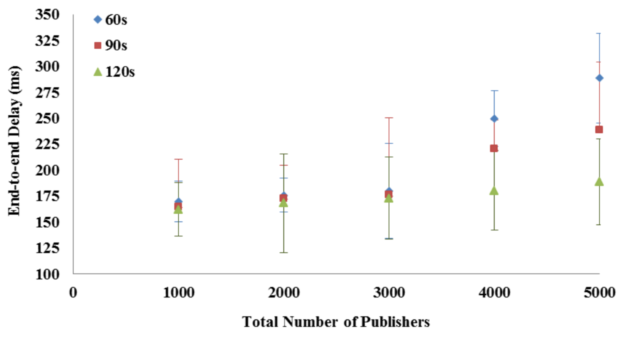

Figure 3. Notification Delay for 2500 Subscribers in Scenario 1

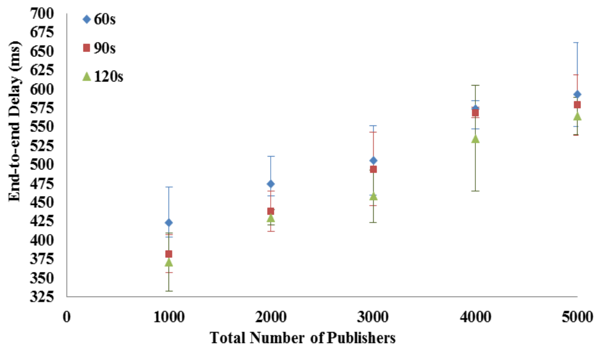

Figure 4. Notification Delay for 5000 Subscribers in Scenario 1

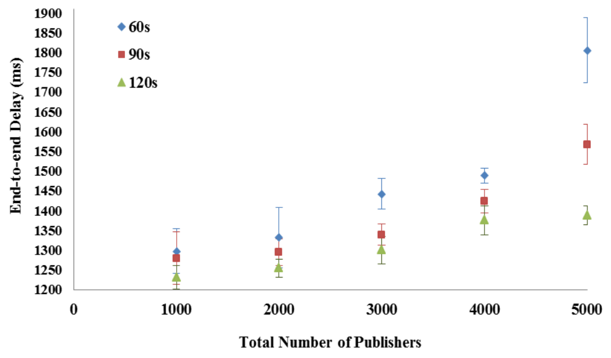

Figure 8. Notification Delay for 10000 Subscribers in Scenario 2

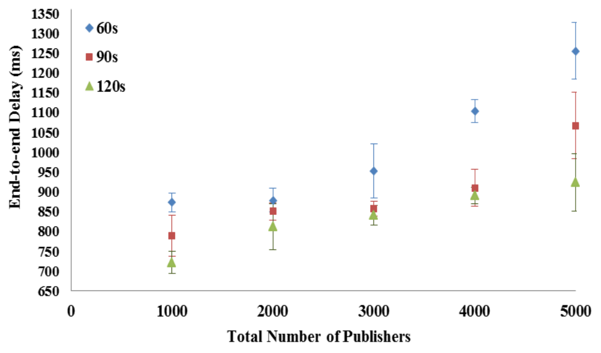

Figure 5. Notification Delay for 10000 Subscribers in Scenario 1

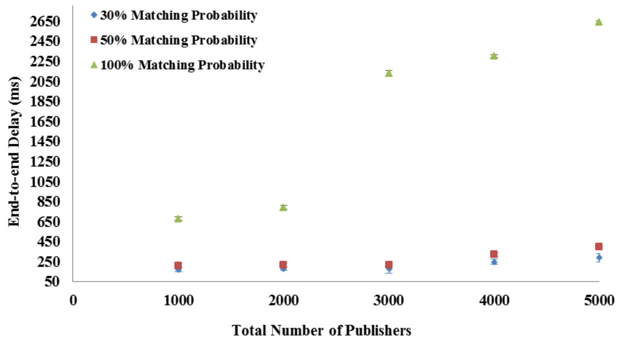

Figure 9. Notification Delay for Different Matching Probability of 2500 Subscribers

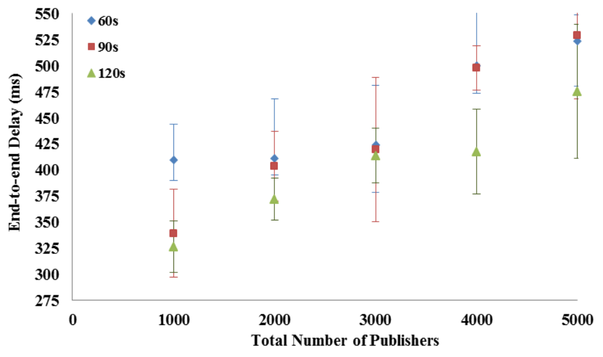

Figure 6. Notification Delay for 2500 Subscribers in Scenario 2

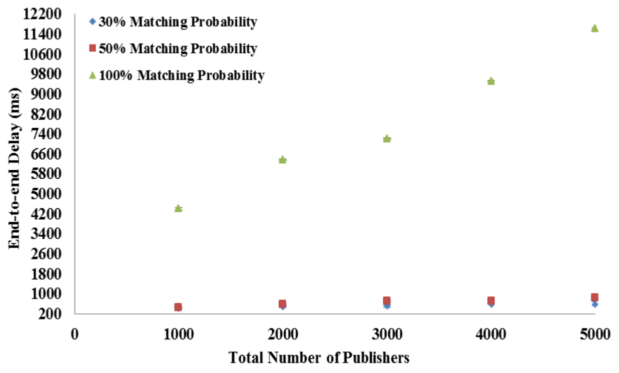

Figure 10. Notification Delay for Different Matching Probability of 5000 Subscribers

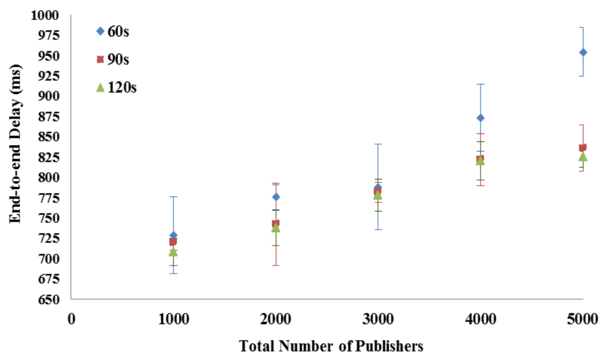

Figure 7. Notification Delay for 5000 Subscribers in Scenario 2

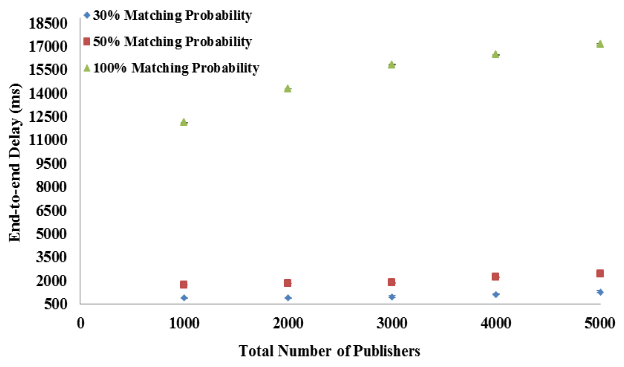

Figure 11. Notification Delay for Different Matching Probability of 10000 Subscribers

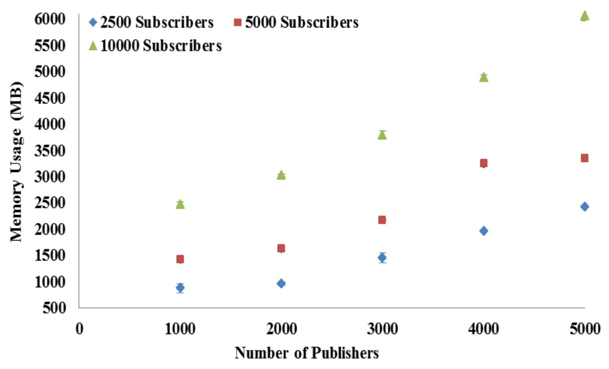

Figure 12. Average Memory Usage by the Brokers

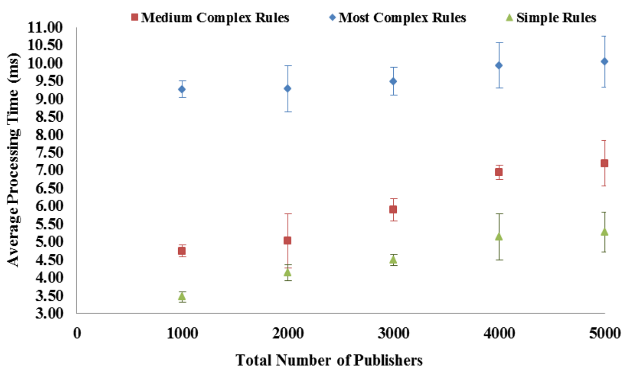

Figure 16. Processing Time for Different Rules of 5000 Subscribers

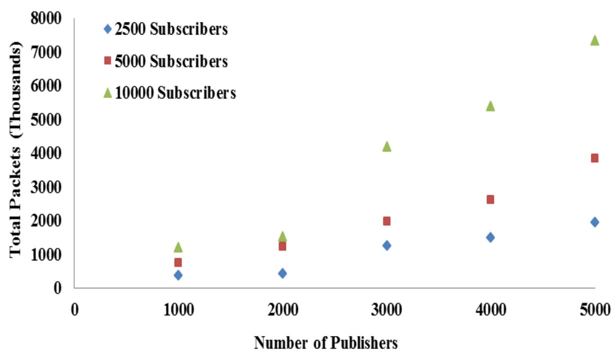

Figure 13. Total Packets Delivery from the Brokers

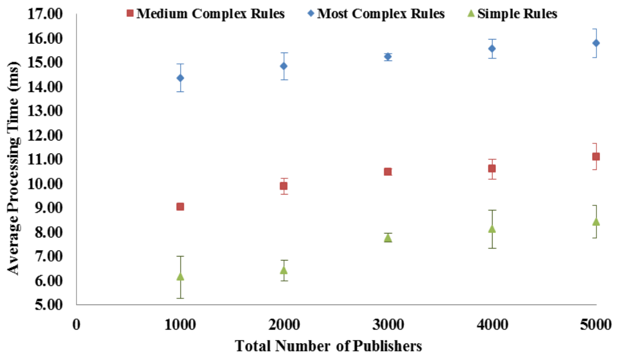

Figure 17. Processing Time for Different Rules of 10000 Subscribers

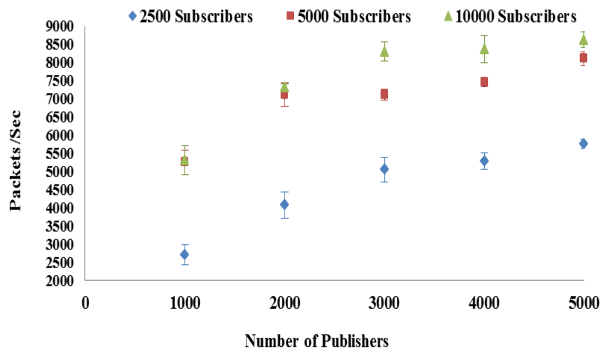

Figure 14. Average Packets Delivery from the Brokers

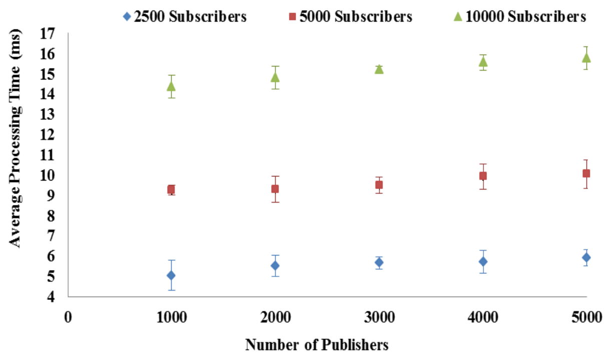

Figure 18. Processing Time for Complex Subscriptions of Different Subscribers

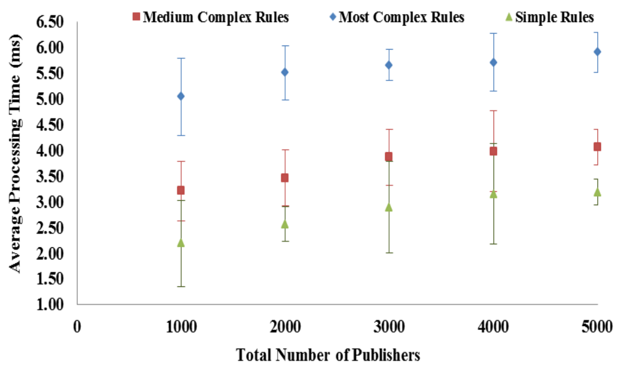

Figure 15. Processing Time for Different Rules of 2500 Subscribers

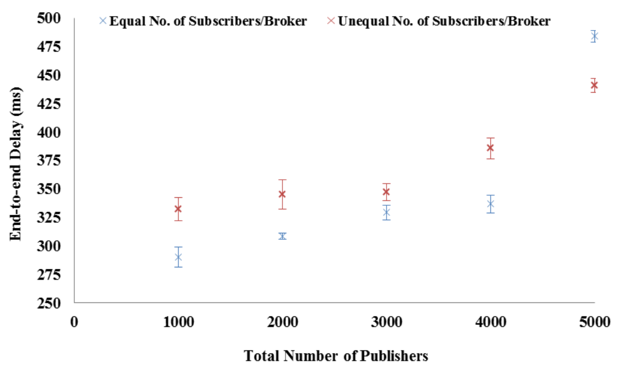

Figure 19. Notification Delay for 2500 Subscribers for different workloads

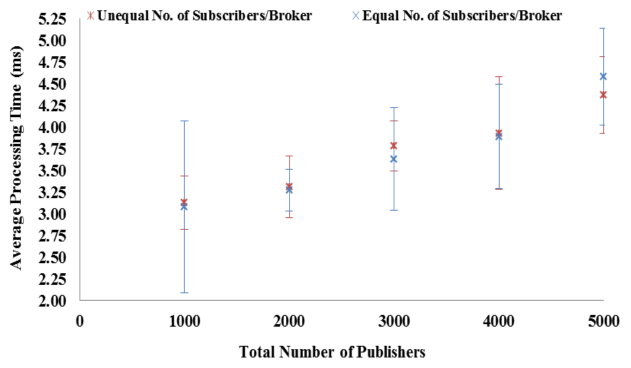

Figure 20. Processing Time for 2500 Subscribers for different workloads

Pongthawornkamol T. et al. [11] present the analysis of a Pub/Sub system over a wireless ad-hoc network with different mobility models of publishers and subscribers. The authors simulate two application scenarios in NS2[4] – "High Traffic Reporting" and "Police Surveillance in Metropolitan Area" and evaluate the performance of ad-hoc routing protocols - Greedy Perimeter Stateless Routing (GPSR)[5] and Ad-Hoc On-Demand Distance Vector Routing (AODV)[6]. This work evaluate the reliability of messages delivery, end-to-end message delivery, and also the load in the node under different mobility models but all the experiments are designed under small numbers of publishers and subscribers that do not ensure the scalability of the system. Moreover, the main focus of this work to check the performance of a Pub/Sub system over an ad-hoc environment rather than over a cloud environment.

Zhang B. et al. [12] provide the idea of moving two Pub/Sub systems – PADRES and OncePubSub over a cloud infrastructure – Xen platform for high performance and scalability. This paper suggests three methods for mapping a Pub/Sub system into a cloud infrastructure – Black-box method, Grey-box method, and White-box method. The Black-box method considers all the brokers as peer servers and utilizes the resources of the VMs, by calculating the performance thresholds and by recommending proper workload distribution among the VMs according to the thresholds. In the white-box method the system is analyzed thoroughly and the brokers that have greater chance of becoming overloaded are identified and mirror of these brokers are created. In the Grey-scale method, the system is set-up such that it can scale up and down automatically that means the VMs starts and shutdowns according to the workload. This work runs a number of experiments and presents the evaluation results for these three different methods and recommends the suitable cases for each deployment. This work is also very close to our work as all the experiments are designed for a cloud infrastructure and a standard IP network. Moreover, the workloads are generated almost in a same way as our case, so that it can match with real-life scenario. In contrast of this work, our work has considered the mobility of the subscriptions. Moreover, our work mainly focuses on the default behavior of OpenStack for deploying real-time applications rather than integrating any methods for load balancing or automatic scale up and/or down, with cloud infrastructure.

[4] http://www.isi.edu/nsnam/ns/

[5] http://www.icir.org/bkarp/gpsr/

[6] http://moment.cs.ucsb.edu/AODV/

V. Discussion

Distributed publish/subscribe system as well as cloud computing are two most prominent and efficient solution for designing an application for ITS with appropriate scalability and reliability while supporting mobility. With this paper we make the following contributions. First, we have implemented real-time public transit tracking application on top of MoPS and evaluated the performance of the system in terms of latency, scalability, and resource consumption. Our experimental results indicate that MoPS can handle large number of subscribers and publishers with proper reliability and negligible notification delay. Moreover, the experimental results strongly indicate that MoPS can handle mobility of the clients efficiently that makes it suitable to deploy an ITS application where automated mobility handling on the system level, and real-time constraints are two most important factors. Second, our fundamental experiment on the performance of the open source cloud management platform, OpenStack for deploying ITS applications. Our experimental results suggest that OpenStack can optimize the workloads' performance by utilizing physical resources efficiently for different workloads on VMs over single physical machine.

As a summary, we can say that MoPS can be deployed in OpenStack to offer an ITS application in a large scale but further investigation is required with large number of brokers which are deployed on multiple physical hosts, located in different geographical locations. Moreover, our current implementation only uses OpenStack as deployment platform; there is clear need to extend our work in order to take the advantage of the cloud environment, that is to scale up or down by allocating or de-allocating VMs automatically when the workload changes. Further, we plan to make our implementation modular, distributed and automated where every component specializes for specific functionalities and distributed over OpenStack VMs and coordinates with each other automatically to serve subscribers. Finally, we would like to evaluate our system with real data set where publications and users mobility represent real scenario of a large city.

Acknowledgment

This research was carried out as a part of the project ITRACT, funded by InterReg IVb NorthSea region programme.

References

[1] Patrick Th. Eugster , Pascal A. Felber , Rachid Guerraoui , Anne-Marie Kermarrec, "The many faces of Publish/Subscribe." ACM Computer Survey, Vol.35, no.2, pp. 114-131, 2003.

[2] Liu Y., Plale B., "Survey of publish/subscribe event systems." In: Indiana University Computer Science Technical Report TR-574, 2003.

[3] Li M., Ye F., Kim M., Chen H., Lei H., "BlueDove: A scalable and elastic publish/subscribe service." In: IEEE International Parallel & Distributed Processing Symposium, pp. 1254–1265. IEEE Press, Anchorage (2011).

[4] Nasim R. and Kassler A. "Distributed Architectures for Intelligent Transport Systems: A Survey." IEEE Second Symposium on Network Cloud Computing and Applications (NCCA 2012), December 3th-4th, 2012, London, UK.

[5] Podnar, I. and I. Lovrek, "Supporting Mobility with Persistent Notifications in Publish/Subscribe Systems." In Proc. Third International Workshop on Distributed Event-Based Systems (DEBS). Edinburgh, Scotland, UK, May 2004.

[6] Mühl G. , Fiege L. , Gärtner F. C. , Buchmann A., “Evaluating Advanced Routing Algorithms for Content-Based Publish/Subscribe Systems” , Proceedings of the 10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, p.167, October 11-16, 2002.

[7] Wen X.,Gu G., Li Q., Gao Y., Zhang X., "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on , vol., no., pp.2457,2461, 29-31 May 2012.

[8] Homepage of OpenStack: http://www.openstack.org/

[9] Ponder I., “Service Architecture for Content Dissemination to Mobile Users”, Doctoral dissertation, University of Zagreb, Faculty of Electrical Engineering and Computing, February 2004.

[10] Google GTFS-realtime feed specification web page: https://developers.google.com/transit/gtfs-realtime/.

[11] Pongthawornkamol T., Nahrstedt K.,Wang G., “The analysis of publish/subscribe systems over mobile wireless ad hoc networks,” in Proc. of ACM MobiQuitous’07, Aug. 2007.

[12] Zhang B., Jin B., Chen H., Qin Z., “Empirical Evaluation of Content-based Pub/Sub Systems over Cloud Infrastructure”, In The 8th IEEE/IFIP International Conference on Embedded and Ubiquitous Computing , 2010.

[13] Corradi, A., Fanelli,M., Foschini, L., “VM consolidation: A Real Case Based on OpenStack Cloud”, Elsevier Future Generation Computer Systems, available online 4 June 2012, DOI: http://dx.doi.org/10.1016/j.future.2012.05.012,2012.