A CLOUD-BASED XML PUBLISH/SUBSCRIBE SERVICE

[1]Chung-Horng Lung, [2]Yang Cao, [1]Mohammed Sanaullah, [1]Shikharesh Majumdar
[1]Department of Systems and Computer Engineering, [2]School of Computer Science
{chlung, majumdar}@sce.carleton.ca, sycao5@gmail.com, mohammedsanaullah11@yahoo.com

Abstract

XML message filtering and routing have been recognized as a standard for data exchange for XML dissemination services. These services are often realized using the Publish/Subscribe (pub/sub) model which is commonly used for various web-based systems. The conventional XML filtering and forwarding technique is an application-layer multicast approach that relies on XML-capable brokers that are built above the network layer for message dissemination. Such an overlay model introduces a great deal of overhead in terms of initial deployment and subsequent operational cost for those XML-capable brokers that are typically supported by geographically distributed Internet Service Providers (ISPs). This paper proposes an approach to the deployment of the XML pub/sub systems over the cloud without the need of any special arrangements of physical brokers across the networks or ISPs). In addition, changes to the software deployed in the cloud can be made directly and easily by the XML pub/sub service provider. The paper describes experiments that use both Amazon and Google clouds for rapid XML pub/sub deployment. In addition, the paper presents a performance comparison between the conventional XML multicast model and the cross-layer model deployed over the cloud.

Keywords: Cloud computing, XML Pub/Sub systems, Amazon EC2, Google compute engine

1. INTRODUCTION

Publish/subscribe (pub/sub) systems have been used in a variety of distributed systems. A message is published by a publisher to subscribers who have submitted their interests covered by those published messages. Pub/sub systems are categorized as centralized and distributed systems. In a centralized system, all the publication messages and subscriptions (or queries) are handled by one single broker which is responsible for performing the tasks of message filtering (message matching) and message routing (message delivery). Once the publication messages are published, they are matched against the queries and are delivered to the matched subscribers. However, if there are a large number of messages, a single broker could be over-utilized to perform the desired operation. This can create a performance bottleneck which can degrade the system efficiency. For a large scale pub/sub system, scalability is one of the most critical issues, since it is being carried out in a wide area network. Use of an overlay model with multiple replicated brokers that are deployed over the Internet is a common approach, especially in the context of XML (eXtensible Markup Language) pub/sub, such as Dai, L. (2010) and Diao, Y. (2003a, 2003b).

XML proposed by Bray, T. (2008) has been considered to be an essential language for representing data and performing information exchange over the web. By exterminating the differences of proprietary protocols between networking, operating systems, and platforms, XML allows different hardware and software components to communicate with each other. A pub/sub system is called a XML pub/sub system if the published messages are in the form of XML format and queries are expressed by XPath or XQuery languages.

XML message dissemination involves the exchange of XML messages between publishers and subscribers. This information exchange takes place through dedicated application-layer brokers which are also known as XML brokers or routers. In XML based pub/sub systems, these brokers identify subscribers that have subscribed queries that match a given publication message. YFilter proposed by Diao, Y. (2003a, 2003b) is the representative technique that was developed for XML message filtering and routing using the multicast technique at the application layer.

YFilter was developed to minimize the number of messages that are sent to all the subscribers. However, this multicast technique requires the deployment of specialized XML-capable routers to build an application-layer topology above the network layer for creating an overlay model. Deployment of the overlay model has high overhead, as special XML-capable brokers or routers need to be deployed over the Internet and to be maintained by service providers. In addition to the specialized brokers, the deployment and maintenance of XML-capable brokers are also time consuming, especially if multiple Internet service providers (ISPs) are involved across multiple networks. Cao, Y. (2011) proposed a cross-layer model for XML message dissemination. When the published messages are transmitted using the cross-layer model, the XML message filtering

time can be reduced compared to the traditional multicast model-based systems, such as YFilter. However, special XML-capable routers still need to be deployed over the Internet with the cross-layer model, which means that high overhead is still inevitable. In short, both the capital expenditure (CAPEX) and the operating expenditures (OPEX) for existing XML pub/sub techniques are high. Further, the deployment delay is also high if multiple ISPs are involved.

The motivation of the research described in this paper is to reduce the cost and time needed for XML-capable router deployment and maintenance over the Internet. In order to achieve the objective, this paper proposes to deploy the XML pub/sub service over a cloud. The benefits of the proposed approach are low cost (for our experiments using the clouds, there was no CAPEX at all for the XML system deployment using the public cloud) and fast deployment (in minutes after system testing) without the need of special XML-capable brokers or services provided by ISPs. Further, with the proposed approach, it is easy to set up the configuration and make changes to the XML broker topology. Another objective of this paper is to compare the performance of two XML pub/sub models over the cloud: the conventional XML multicast model used by Dai, L. (2010), Diao, Y. (2003a, 2003b) and the cross-layer model proposed by Cao, Y. (2011). These two models will be explained further in Section 2.

In Lung, C. (2014), the authors proposed a novel cloud-based XML pub/sub system and demonstrated the benefits of fast deployment of the proposed approach over the cloud for XML multicast model. This paper is an extension of that paper. The main contributions of this paper include: extension to two different cloud providers for XML pub/sub systems, through experiments performed on a real system, and a thorough performance comparison of two different XML pub/sub models – multicast and cross-layer – deployed over the cloud.

The rest of the paper is organized as follows: Section 2 highlights the related work. Section 3 describes our approach using the cloud for XML services. Section 4 presents the experiments and some results. Finally, Section 5 is the summary and future research directions.

2. RELATED WORK

The idea of pub/sub services in the cloud is not totally new. For instance, Amazon Simple Notification Services (SNS) (2016) supports topic based pub/sub service in the cloud, and Lublinsky, B. (2012) also described an approach to support pub/sub services based on Amazon Web Services (AWS). Li, M. (2011) presented an approach called BlueDove for pub/sub services in the cloud based on Cassandra. BlueDove supports attribute-based pub/sub systems in the cloud. Ma, X. (2013) discussed attribute-based pub/sub systems in the cloud. In attribute-based pub/sub systems, partitioning of the content space into sub-spaces is performed and a set of matchers are organized into an overlay.

However, to our knowledge, no papers have addressed XML pub/sub systems in the cloud, which is the main concern of this paper. In other words, none of the existing works has considered the deployment of an XML pub/sub service on a cloud.

User queries and publication messages expressed in XML have a high degree of flexibility in terms of the message structure and semantics. But the complexity and processing time of the XML message filtering and matching operation and the XML query aggregation operation could be high. As an example, Yfilter proposed by Diao, Y. (2003a, 2003b) processes an XML publication message by searching from the root of a non-deterministic finite automata (NFA) corresponding to a group of aggregated queries. Yfilter then tracks all matched transition states for each start tag. There is a selection operator for each accepting state to handle the value-based predicate in the corresponding path. For a complex tree-structure XPath query, Yfilter splits such a query into a set of multiple simple path expressions path by path.

Further, Li, M. (2011) and Ma, X. (2013) considered cloud deployment and performance comparison over the cloud. The model consists of a front-end layer and a back-end layer. The front-end layer consists of dispatching servers that send subscriptions and messages to matching servers at the back-end, while the back-end layer is a set of matching servers that perform message matching and message delivery to subscribers.

The architecture model adopted in Li, M. (2011) and Ma, X. (2103) is different from the cross-layer model proposed by Cao, Y. (2011). The cross-layer model consists of publisher edge (PE) and customer edge (CE) servers or brokers to perform filtering of publication messages and aggregation of queries. The PEs receive publication messages from the publisher and then perform the filtering operation to identify those CEs that are connected to the matched subscribers. The CEs are responsible for aggregating queries received from the subscribers, forwarding aggregated queries to PEs, and matching publication messages with local subscribers and delivering messages to local subscribers. The intermediate nodes across the network are simply responsible for forwarding messages. Section 2.2 describes the model in more detail.

The following subsections briefly explain XML message dissemination and the two architecture models for XML pub/sub systems that are considered in this paper – traditional multicast model and the cross-layer model.

2.1 XML MESSAGE DISSEMINATION

XML message dissemination or XML message filtering and routing is a main task for an XML broker or router. To deliver messages to a specific group of subscribers, the XML filtering task is to parse the XML publication

messages and identify the matched XPath queries (and the corresponding subscribers) that have been received at the broker.

Several XML filtering techniques have been reported in the literature: Yfilter in Diao, Y. (2003a, 2003b), Afilter in Candan, K. (2006), Gfilter in Chen, S. (2008), and Bfilter in Dai, L. (2010), Hfilter in Sun, W. (2008), and Pfilter in Saxena, P. (2013). For brevity, the paper only focuses on YFilter, as it is the representative approach in the field, but our approach is not limited to a particular XML filter technique. YFilter is an automata-based filtering algorithm, which is explained further in the next sub-section.

2.2 XML PUB/SUB BROKER ARCHITECTURE

The key functional components of an application-layer XML pub/sub system include: (a) a filtering engine which receives publication messages from the publisher and identifies subscribers that have sent the subscriptions or queries; (b) a query aggregation component that groups related queries to reduce the number of queries and forwards the aggregated queries to its upstream broker(s); and (c) a routing and message delivery component which constructs and maintains the network topology for queries and sends publication messages to the neighboring brokers and/or subscribers.

The application-layer XML multicasting technique has been proposed by Carzaniga, A. (2001), and adopted in Siena, Yfilter, Afilter, Gfilter, and Bfilter, etc. In such a model, as shown in *Figure 1*, each broker stores query information and performs query aggregation and publication message filtering operations hop by hop.

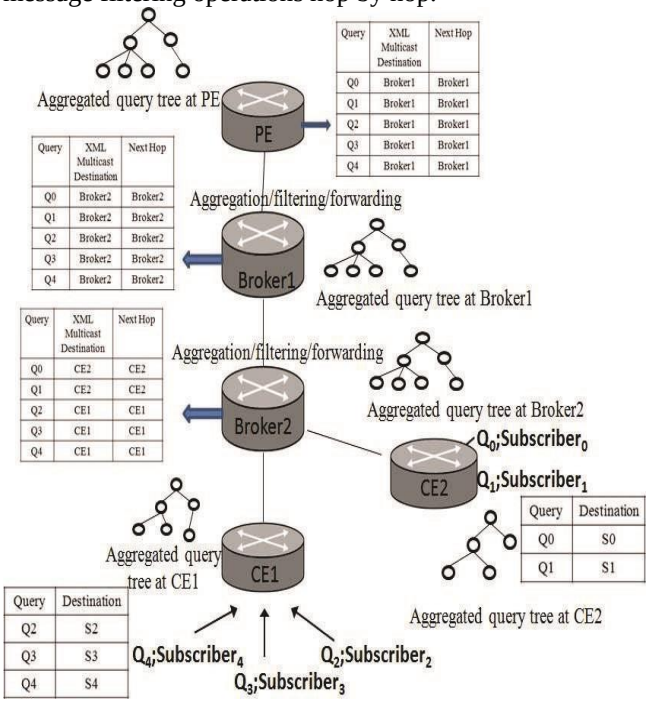

Figure 1. XML cross-layer model in Carzaniga, A. (2001)

For query processing and aggregation, the application-layer multicast model consists of the following main steps:

1) Queries from subscribers are sent to proximity customer edge (CE) brokers.

2) Queries received at CEs will be aggregated and forwarded to the upstream broker(s).

3) On receiving the aggregated queries from connected CEs, intermediate brokers construct the routing table and then in turn forward the aggregated query and routing information to the upstream broker(s). This step will be repeated for all intermediate brokers until the aggregated query reaches the PE broker which is connected to the publisher. The PE also creates the XML routing table for subscribers based on the queries it has received from downstream brokers.

For publication messages, the steps for the application-layer multicast model include:

1) The publisher sends publication messages to PE(s).

2) PEs and all intermediate brokers perform the publication message filtering operation and identify the matched subscribers. Publication messages will be forwarded to downstream brokers (intermediate brokers or CEs) that have information for the matched subscribers.

3) When CEs received publication messages, CEs also identify matched subscribers via the filtering operation and forward the publication message directly to the subscribers.

According to the process, each broker in this model needs to perform both query aggregation and filtering/routing operations, which results in high delay, especially if the number of brokers or the network size is large.

The cross-layer model, as presented in *Figure 2*, was proposed in Cao, Y. (2011) with an aim to reduce the aforementioned processing overhead or delay at each broker. In this model, intermediate brokers, e.g., pBroker1 and pBroker2 as shown in *Figure 2*, are regular network devices. In other words, the intermediate brokers only perform the message forwarding functionality like normal Internet routers which forward messages, and no XML query aggregation and message filtering operations will be performed at intermediate brokers. In the cross-layer model, query aggregation and publication message filtering operations are only performed at edge brokers, e.g., CEs and PEs, which relieves the intermediate brokers, pBroker1 and pBroker2, from performing the same tasks repeatedly. The existing filtering algorithms, such as Yfilter, Afilter, Gfilter, Bfilter, Hfilter, and Pfitler, can still be used in the cross-layer model at PEs and CEs.

The number of publication messages delivered to subscribers is optimal using the conventional multicast model in Diao, Y. (2003a, 2003b), as the query aggregation operation is conducted at each broker and the nature of multicast for publication messages. However, each XML broker needs to perform both query aggregation and

publication message filtering operations, which have high computational overhead or delay. Compared to the conventional multicast model, the cross-layer model may forward a message multiple times, since the model makes use of the unicast scheme from a PE to differnet CEs. However, the number of unicast messages is small, as the unicast messages are only destined to CEs, not the end subscribers. Therefore, the processing time can be significantly reduced as no aggregation or filtering operation needs to be carried out at intermediate brokers.

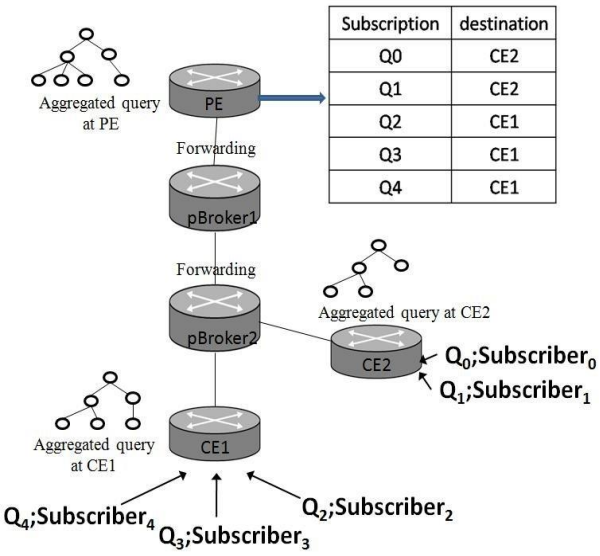

Figure 2. XML cross-layer model in Cao, Y. (2011)

2.3 Cloud Computing, Amazon EC2 and Google Compute Engine

Cloud computing has received a great deal of attention in recent years, as it provides a cost-effective way to meet the changing business needs. National Institute of Standards and Technology (NIST) defined cloud computing as "*a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" (2012). Using cloud technologies, organizations can efficiently scale their IT infrastructure, such as hardware, software and services, effectively via cloud service providers instead of investing time and money on the on-premise infrastructure.

Two of the popular cloud services are Amazon EC2 (2016) and Google Compute Engine (2016); both provide a cloud computing platform that offers infrastructure and application services. With the cloud service, the user (such as enterprises) can run virtually any kind of applications in the cloud. In addition, both cloud services allow users to create scalable applications for high demands and workloads without investing upfront time and money (CAPEX) on the actual infrastructure which is time consuming and costly.

Both EC2 and Google Computer Engine provide elastic computing resources depending on the demands. Additional new instances can be set up or existing instances can be relinquished in a short period of time (in minutes). Users can access remote virtual servers running in a data center. Each virtual server is called an "instance". Users can request multiple or any number of virtual servers as needed online. Moreover, in the context of XML pub/sub system deployment, it means that no special XML-capable brokers need to be set up by ISPs over large geographically separated areas, which can be translated into tremendous savings in time and cost.

3. XML Pub/Sub Services via Clouds and Experiment Setup

As described in Section 2.2, XML pub/sub service using the conventional multicast model gives rise to a high delay for query aggregation and message filtering operations that are performed at each broker. The cross-layer model mitigates the problem by removing these expensive operations from the intermediate brokers. However, both techniques require dedicated or special XML-capable brokers or routers. For the multicast model, each broker needs to be XML-capable, while for the cross-layer model, only the edge brokers, e.g., PEs and CEs, need to be XML-Fcapable and the intermediate brokers just maintain the state for the multicast tree and perform the message forwarding operation. In either case, deployment of special XML-capable brokers requires initial CAPEX investment and it is time consuming to make the services operational. Moreover, multiple ISPs are needed across the Internet.

The approach described in this paper proposes to deploy the XML pub/sub services over the cloud. The primary motivation is to validate if XML pub/sub services can be rapidly and cost-effectively deployed over the cloud. The rationale behind this is that only the XML broker software system needs to be installed on virtual servers and the configuration of a logical network for the virtual servers (either as XML brokers for the conventional multicast model or edge brokers for the cross-layer model) needs to be performed. Moreover, as will be discussed in the next sub-section, the deployment does not need real physical XML-capable brokers and is not involved with any ISPs, even if two logical XML brokers (software systems) are far away in different parts of a country or in the world. On the other hand, the minimum initial cost to deploy the physical brokers using the existing approaches (e.g., overlay multicast layer and the cross-layer models) includes the cost for the application layer network infrastructure established by ISP(s) as well as XML-capable brokers. Further, in this

case, those physical brokers typically do not support elastic demands.

This section describes three experiments. The first experiment is for investigating the feasibility of deploying XML pub/sub services over the cloud and the efficiency of the deployment. The second experiment is performed to compare the performance of the traditional multicast model and the cross-layer model when deployed over the cloud using Amazon's EC2 instances in different datacenters around the world. The third experiment is to validate the proposed cloud-based XML pub/sub using both Amazon and Google cloud services. The objective is to validate interoperability and ease of deployment using multiple cloud providers for the pub/sub service.

3.1 Launch Instances on Amazon EC2 and Google Compute Engine

Amazon EC2 is a cloud from Amazon WS that provides Infrastructure as a Service (IaaS). Amazon provides pre-configured Amazon Machine Images (AMIs) which are templates of cloud instances. Google Compute Engine is an IaaS provided from Google. Both Amazon EC2 and Google Compute Engine provide instances of various computing capacities and various types, with various memory and CPU capacities for example. Users can upload applications to Amazon EC2 or Google Compute Engine instances rapidly and easily.

By comparing the steps for launching Amazon EC2 instances and Google Compute Engine instances, we find that the steps are very similar and are described next. Note that automatic launching of cloud instances can be achieved using scripts, e.g., Python.

1) Log in Amazon Web Service Console or Google Cloud Platform
2) Define instance type, instance name, project ID, specify disk image, generate public/private key, network security rules, and select zones
3) Create a new instance
4) Launce a new instance
5) Instance details (such as public IP, public DNS, and launching time) are shown in the console after the status of the newly launched instance becoming "running".

After launching cloud instances, we installed Java SDK 7.0 and Eclipse on the instances. Then, our XML router prototype is uploaded to the cloud instances. After that, the logical network configuration is set up on each cloud instance.

3.2 Experimental Setup

This section describes the steps in setting up the test-bed for two pub/sub models: traditional multicast model and the cross-layer model, over the cloud.

To run the experiments using the cloud services offered by Amazon, virtual machines (VMs) (instances of XML brokers) with an Amazon M1.medium configuration was selected,. Such a VM consists of 16GB RAM, 2 ECUs (EC2 Compute Units), 3.75 GB memory, 410 GB of disk storage, and an 100 Mbps Ethernet of. An ECU2 is equivalent to a 1.7GHz Intel Xeon processor (2016).

Another use case that consists of brokers being deployed over two cloud providers is considered. *Table 1* describes the configurations from the respective service providers used in our experiments. This use case is considered for three reasons: first, no cloud providers may cover all the required or desirable geographical locations; second, the cost could be a factor for selecting a cloud provider, as the pricing may be different for cloud providers; and third, increased reliability for the pub/sub provider, as it can be connected to more than one cloud provider. The concept is similar to multi-homing in Troan J. (2014) that has been used to connect to multiple ISPs. An experiment using two cloud providers has been conducted to evaluate the interoperability between cloud providers as well as the ease of deployment over multiple cloud providers.

Table 1. Amazon EC2 and Google compute engine VM instances description for Topology #3

Cloud IaaS Service Provider	Description
Amazon EC2	t2.micro (1 vCPU, 1GB memory, Microsoft Windows Server 2012 R2 Base)
Google compute engine	g1-small (1 vCPU, 1.7 GB memory, Red Hat Enterprise Linux 6.6 x86_64)

Three network topologies have been experimented with. The first is a simple network for feasibility study and validation of the deployment of the XML pub/sub system over the cloud (see the description provided in the next paragraph). The second network topology consists of brokers deployed over the cloud in different areas of the world, but all the brokers belong to the same cloud provider: Amazon. The third network topology not only consists of brokers in various geographical locations of the world, but also uses two cloud providers: Amazon and Google.

For the second and the third network topologies, time of day is an important factor for performance evaluation, as traffic loads for different time zones in different areas of the world will affect the results. The network topology and time of day are discussed further as follows.

Topology #1 includes two brokers: a PE broker in the USA East and a CE broker in the USA West. This simple network topology is considered for the feasibility study in terms of functional requirements and the efficiency of the deployment of the XML pub/sub services over the cloud. Since the number of brokers is minimal, i.e., only one PE and one CE are used, the end-to-end delay is expected to be identical for both the multicast and the cross-layer models,

as edge brokers for both models perform the same operations.

Topology #2 includes four brokers with each broker located in a different geographic region. This topology is organized as follows: PE broker A is located in Richmond, Virginia (USA East), an intermediate Broker B in San Francisco (USA West), another intermediate Broker C in Dublin Ireland and the CE broker D in Singapore (Asia Pacific region). The time zones for all the brokers are different; hence the traffic loads may be very different at those four different locations.

Topology #3 includes seven brokers in total that are offered by two cloud providers: Amazon and Google. *Figure 3* depicts the topology, including the cloud providers and the location of the brokers. Broker A is the PE broker and brokers E and G are the CEs brokers. Using multiple cloud providers can be beneficial for some services based on proximity for lower propagation delay and/or the reliability reason in case the service from one cloud provider fails.

Time of Day: The time of day has an important effect on the performance results of the system due to the variability in traffic patterns observed during different times of a day. As indicated earlier, the brokers deployed over the cloud are placed in different geographical regions, each of which is characterized by a different time of the day. At a given point in time, each broker is thus subjected to different intensity of Internet traffic which is closely related to the respective time of day and will affect the network delay. Hence, two different scenarios related to time of day are considered to mitigate the factor of different traffic patterns. This leads to two diverse scenarios with high and low traffic patterns due to the fact that those regions may have a time difference up to 12 hours.

Scenario #1, the Internet traffic for USA East and USA West regions is low (night time traffic for example), and traffic for Ireland (Europe) and Singapore regions is high (for day time traffic for example).

Scenario #2, the traffic for USA East and USA West is high, and traffic for Ireland and Singapore is low.

The connection between two instances of brokers is realized using Amazon "Elastic IP" described in Amazon (2016). Each Amazon instance is associated with a static IP address referred to as "Elastic IP" which is designed for dynamic cloud computing. For any zone failure, these elastic IP addresses can be remapped to any of the other or additional new instances. Such a system is more robust than the IP address system used in the.

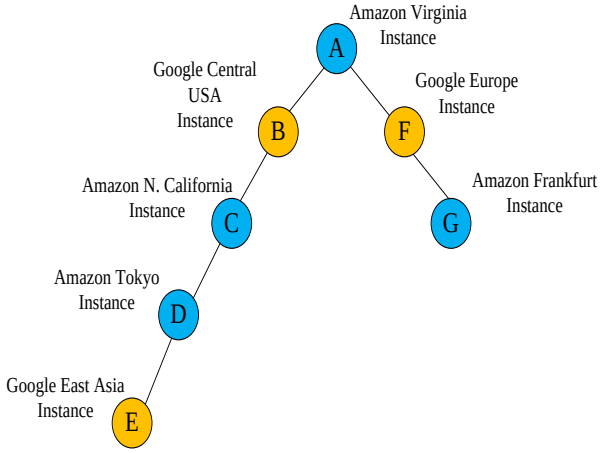

Figure 3. Network Topology #3 with Amazon and Google cloud services

3.3 Experimental Parameters and Metrics

A number of system parameters were considered for the experiments. *Table 2* lists those parameters and the values of these parameters used in experiments. A single parameter is varied in each experiment while the other parameters are held at their default values which are highlighted in bold italic in *Table 1*. Parameters with a single value are fixed parameters. Default values are based on parameter values used in the existing research works described in the

Table 2. Parameters and Values Used for Experiments

Parameters	Descriptions	Values
Message type	Test message type	Recursive messages with book.dtd
Message size	Test message size (in Kbytes)	{1, 5, ***10***, 15, 20}
λ	Message arrival rate (in messages/sec)	{0.5, ***0.85***, 2, 4, 8}
Number of XPath queries	Number of queries or subscriptions	***20,000***
Prob(//)	Probability that an XML element has //	***0.2***
Prob(*)	Probability that an XML element has *	***0.2***
Prob(branching)	Probability that a query has branches	***0.1***
Prob([ ])	Probability that a query has predicates	***0.2***
Query depth	The longest path length of a test query tree	***6***
Match ratio	The ratio of matched queries over test query population	**≈ Around {*0.05***
Query duplication	If duplicated queries are allowed?	***Allowed***

literature of Diao, Y. (2003), Chen, S. (2008), Chand, R. (2008), and Cao, Y. (2011).

The message type used in the experiments is *book.dbd* which was also used in Afilter by Candan, K. (2006), and Gfilter by Chen, S. (2008). For message size and the message arrival rate (λ), the default values are 10KBytes and 0.85 message/sec, respectively.

Several metrics have been used for performance evaluation: average filtering time, average processing time, average round-trip time, and average end-to-end (E2E) delay. But E2E delay is the most important one in comparing the performance of the conventional multicast model and the cross-layer model deployed over the cloud. A number of such key parameters are defined next:

Average filtering time: The filtering time is the time taken between the times at which a publication message enters and leaves the filtering engine (and is sent to the next broker) in a broker. The average filtering time is the average value obtained from 100 experimental runs.

Average processing time: The average time needed for a message that is received till the message is sent by a broker for 100 publication messages. The processing time includes the XML message filtering time and the overhead of processing the message.

Average end-to-end (E2E) delay: E2E delay is the time taken for a message to traverse the network from the source (PE) to the destination (CE). E2E delay includes the transmission delay, and processing delay and queuing delay for intermediate brokers across the networks. The average results are calculated based on values obtained from 6,000 experimental runs, which may last 4 to 5 hours.

In our measurements, each message is provided with a time stamp (t_1) before it is sent by the PE. Since two machines (source and destination) are located at different regions, their clock times are expected to be different. In order to handle the mismatch in clocks, an acknowledgement (ACK) message is sent back from the destination (CE) to the source (PE) which assigns another timestamp (t_2) to the ACK message. Thus, with both time stamps are provided by the same machine, the difference between these two time stamps ($t_2 - t_1$) is used as the Total Delay of a message from PE to CE and the time required for the small ACK message from CE to PE. The E2E delay is then calculated as:

E2E delay = Total Delay – (RTT/2)

where RTT is the round trip time obtained from the ping command from PE (source) to CE (destination). The estimated one way propagation delay is assumed to be half of the RTT (or RTT/2). RTT/2 is used here to obtain the approximate one-way propagation delay from the source (PE) to the destination (CE), since the processing time for the ping command and the ACK message is negligible. Hence, the difference between Total Delay and RTT/2 is the approximate E2E delay along all the brokers from PE to CE, including transmission delay and queuing delay across the networks, and the processing delay at each broker. Average E2E delay is the average results of 100 runs for Section 4.1 for initial evaluation and 6,000 runs for Section 4.2 for higher accuracy.

4. Performance Results

In this section, we compare the experiemntal results and analyze the performance of different XML pub/sub models using Amazon EC2 instances and Google Compute Engine instances.

4.1 Results on Amazon EC2

This section highlights the experimental results performed with Amazon EC2. As stated in Section 3, Topology #1, is used to investigate the feasibility of providing an XML pub/sub service using a cloud and the efficiency of such a deployment. Topology #2, is used to demonstrate the deployment of XML pub/sub systems using multiple distant brokers spanning diverse regions. In addition, this topology is also deployed to compare the performance of two models over the cloud.

For Topology #1 where only one PE and one CE are used, each broker instance can be set up properly in the cloud environment within minutes without the assistance from any ISP or cloud service provider. Moreover, there is no need to set up an application-layer XML pub/sub system on top of the network layer. In other words, deployment of XML pub/sub services over the cloud can be completed rapidly. On the contrary, in a traditional non-cloud case, setting up one broker in the east and one in the west of USA requires the services from one or multiple ISPs. With such a cloud deployment, the time and cost can be dramatically reduced, as the deployment is short (only minutes) and no CAPEX for the network services is needed. The OPEX is also much lower because services from ISP(s) are not needed.

As expected, for Topology #1, the performance results are almost identical for the conventional multicast model and the cross-layer model deployed over the cloud. This is because the PE and CE for two models have identical operations for this simple scenario. *Table* 3 shows the processing time for both models, while *Figure 4* demonstrates the average E2E processing delay.

Table 3. Average Processing Time (ms) for Edge Brokers for Topology #1

Message Size	Multicast	Cross-Layer
1 KB	7.70	7.86
5 KB	24.64	24.75
10 KB	41.67	41.58
15 KB	57.41	56.21
20 KB	85.20	84.80

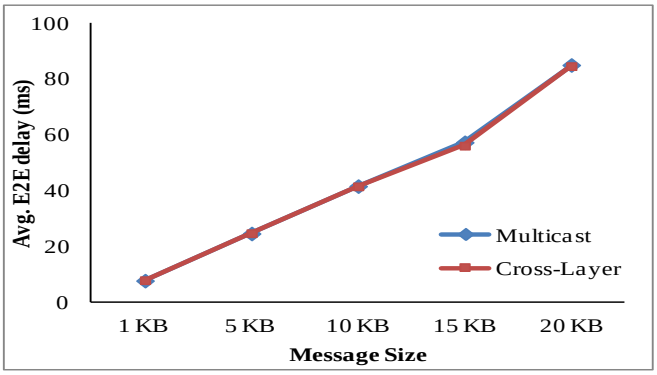

Figure 4. Average E2E delay (ms) for Topology #1

Effect of Message Size. For Topology #2, *Table 4* shows the processing time for various message sizes for intermediate brokers B and C. *Figure 5* depicts the average E2E delay for different message sizes for Scenario #1, as described in Experimental Setup.

Table 4. Average processing time for intermediate brokers for Topology #2

Message Deliver Models	B	C
Multicast	45.70	44.46
Cross-Layer	2.52	2.03

As illustrated in *Figure 5*, for a given message size, the average E2E delay for the cross-layer model is significantly lower than that of the multicast model. This shows the advantage that accrues from no filtering operation at intermediate brokers for the cross-layer model. Similar results were observed for Scenario #2; hence, it is not presented here.

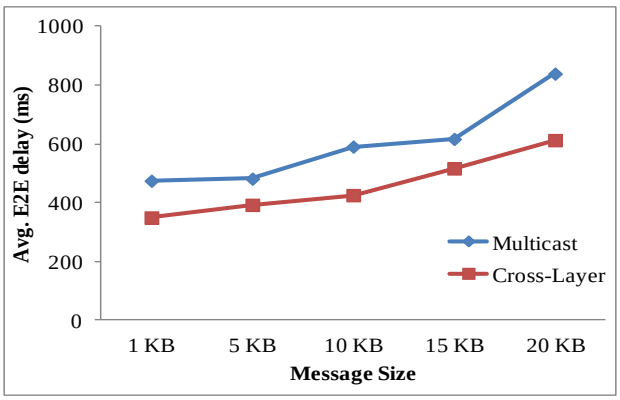

Figure 5. Average E2E delay(ms) with respect to message size for Topology #2 and Scenario #1

Effect of Arrival Rates. The next set of experiments is related to the publication message arrival rates. *Figure 6* and *Figure 7* demonstrate the results.

For *Figure 6* and *Figure 7*, the average E2E delay is determined with different publication message arrival rates, while keeping all other parameters at their default values as listed in *Table 2*. The arrival rate, λ, is varied from 0.5 msg/sec to 8 msg/sec.

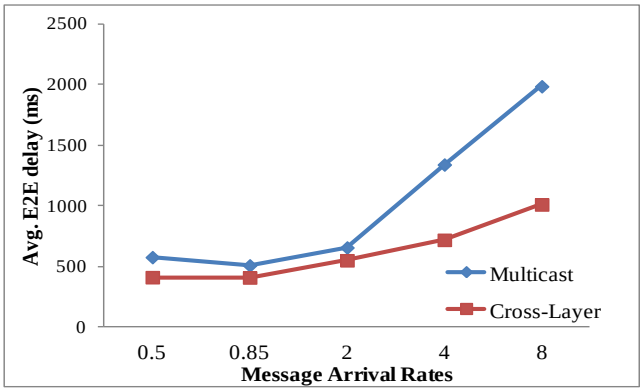

Figure 6. Average E2E delay (ms) with respect to message arrival rate for Topology #2, Scenario #1

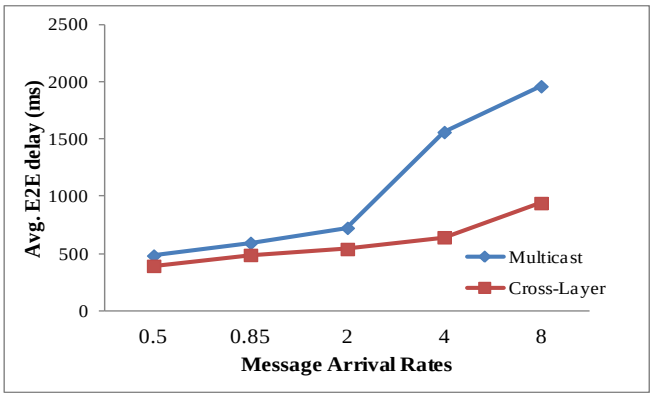

Figure 7. Average E2E delay (ms) with respect to message arrival rate for Topology #2, Scenario #2

For any given arrival rate, the cross-layer model demonstrates a superior performance in comparison to the traditional multicast model. As shown in both figures, the average E2E delay for the multicast model increased sharply with an increase in λ, whereas the cross-layer model demonstrated a significantly lower rate of increase in the average E2E delay in response to the change in the arrival rate. The trend of the performance results for both models is similar to that reported in Cao, Y. (2011) that was conducted in a local area network (LAN) environment. This indicates that, for these two models, the performance trends observed on a cloud-based system agree with those observed on a non-cloud-based environment.

4.2 Results Using Both Amazon EC2 and Google Compute Engine

This section presents experimental results for systems that use a mix of both Amazon and Google cloud services at the same time. The main objective is to investigate the interoperability of these two major cloud providers in deploying the pub/sub system and the ease of such a deployment. With multiple cloud providers, the pub/sub service provider may choose specific locations that are beneficial to the provider or users, e.g., network size, proximity of datacenters, performance, and cost. Further, reliability of services can be improved with multiple cloud providers in case the service from a cloud provider is interrupted.

For the experiment, only the default values of the system parameters, as shown in *Table 2*, were chosen to demonstrate the feasibility of deploying pub/sub services over multiple cloud providers at the same time. The results were average of 6,000 runs for a publishing rate of 0.85 msg/sec. The results presented in *Figure 8* not only illustrate that the cross-layer model has better performance, but the trend also is consistent with that observed when using only the Amazon cloud service. Note that the brokers deployed over different clouds also communicate through the public Internet at the network layer. Consequently, the results may have high variations due to the fluctuation of the public Internet traffic.

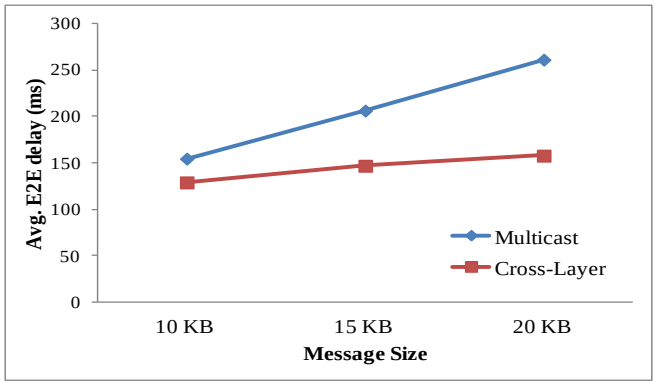

Figure 8. Average E2E Delay (ms) with respect to Message Size for Topology #3

We have compared the performance of two different message delivery models for XML pub/sub systems using two different cloud service providers. In our XML router prototype, a network configuration file records mappings between broker names and public IPs. Therefore, there is no deployment difference for our XML router prototypes that are deployed on different cloud service providers. Moreover, there is no difference in the procedure for deployment of our XML router prototype on the LAN environment used by Cao, Y. (2011) with that used on the cloud environment.

5. Discussions and Conclusions

XML pub/sub systems have been studied widely. The traditional XML pub/sub system builds an overlay application multicast layer which sits above the network layer. The overlay multicast layer requires the deployment of special XML-capable brokers which could take a long time, especially when multiple service providers are involved. Further, both the initial CAPEX and subsequent OPEX will be high, depending on the size of the network and the network topology.

Pay-per-Use and elastic services are two of the main advantages for cloud computing. This paper presents an approach of moving the XML pub/sub services to the clouds, which can eliminate the need for deploying special XML-capable brokers across possibly different networks that may even be administrated by multiple ISPs. In addition, there is no need to rely on specific pub/sub services offered by an ISP or multiple ISPs. Hence, both time and cost can be significantly reduced.

The proposed approach was validated by deploying existing XML pub/sub systems over Amazon cloud services, and a combination of Amazon and Google cloud services. The experiments of the XML pub/sub services were deployed efficiently in a short period of time. For our systems using the cloud, the CAPEX was zero, e.g., no additional cost is needed to deploy the XML broker system on the cloud. On the other hand, OPEX is determined by the usage of the cloud services. For the M1.medium configuration offered by Amazon EC2 that was used in our experiments, the cost is $0.120/hour for our selected configurations in Amazon Elastic Compute Cloud (EC2) (2016). *Table 5* and *Table 6* present some specifications of Amazon EC2 (2016) and Google Compute Engine (2016) instance types and cost, respectively. OPEX is expected to be lower than the non-cloud deployment, as specific pub/sub services are needed for the traditional non-cloud multicast model. For our experiments using two delivery models (multicast and cross-layer) on multiple instances (e.g., three Google Compute Engine instances and four Amazon EC2 instances for Topology #3) with different parameter values (three message sizes and the default arrival rate at 0.85 publication message/sec), and 6,000 runs for each data point, the total cost was about $320.

Table 5. Some Amazon on-demand instance type (Windows) specifications

Instance type	Platform	vCPU	Memory (GB)	Cost/hour ($)
t2.Small	64-bit	1	2	0.036
t2.Medium	64-bit	2	4	0.072
m3.Large	64-bit	2	7.5	0.266
m3.xLarge	64-bit	4	15	0.532

If considering the elasticity feature of cloud computing, the proposed approach will be even more advantageous than the traditional non-cloud-based approach, as the cloud

Table 6. Some Google compute engine instance type specifications

Instance type	Platform	vCPU	Memory (GB)	Cost/hour ($)
f1-micro (windows server image)	64-bit	1	0.6	0.02
g1-small (windows server image)	64-bit	1	1.7	0.02
f1-micro (red hat enterprise linux images)	64-bit	1	0.6	0.06
g1-small (red hat enterprise linux images)	64-bit	1	1.7	0.06

environment supports scaling up/down services. Using the non-cloud-based approach, typically either over-provisioning of broker capacities or deployment of additional brokers is needed for higher demands as well as the cost for pub/sub services. In either case, the cost will be significantly higher than using the cloud, as the elasticity can be effectively utilized based on demands. The proposed approach hence is cost effective and is particularly useful for small and medium enterprises (SMEs) with financial constraints.

A comparison of two XML pub/sub models – the traditional XML multicast model and the cross-layer model – has been performed. First, both models can be deployed in the cloud efficiently, even across multiple cloud providers. Next, the results show that the performance for XML publication message dissemination and delivery is considerably lower for the cross-layer model proposed in this research in comparison to the traditional multicast model using cloud computing. This is observed to be in line with previous studies using non-cloud-based approaches Cao, Y. (2011).

In summary, based on our experiments, the advantages of the XML pub/sub services deployed in the cloud include:

1) Easy installation and short deployment time of multiple brokers across diverse geographical locations;
2) Support of interworking between cloud providers and hence support of flexibility of broker locations and topology administrated by different cloud providers;
3) Quick modifiability for the network topology;
4) Auto-scaling for controlling the resources using the cloud features;
5) Increased reliability with multiple cloud providers in case of failures;
6) Lower costs in terms of both CAPEX and OPEX.

A short discussion of directions for future research is presented. Analyzing the performance of the proposed technique using the real XML pub/sub workload forms an interesting direction for research. The analysis of the cost savings, including both CAPEX and OPEX, based on real system workload and demands is worthy of investigation. For a large network, a tree-based or a mesh network topology may be needed to connect a large number of brokers Systems with such models warrant investigation.

6. Acknowledgments

The authors would like to thank Alcatel-Lucent, Ontario Centres of Excellence, and Natural Sciences and Engineering Research Council of Canada, for supporting the research.

7. References

Amazon Elastic Compute Cloud (EC2) (2016). Retrieved March 28, 2016, from http://aws.amazon.com/ec2/.

Amazon Simple Notification Service (2016). Retrieved March 28, 2016, from http://aws.amazon.com/sns/.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., and Yergeau, F. (2008). Extensible Markup Language (XML) 1.0. World Wide Web Consortium, Recommendation, from http://www.w3.org/TR/2008/REC-xml-20081126/.

Candan, K., Hsiung, W.-P., Chen, S., Tatemura, J., and Agrawal, D.(2006). Afilter: Adaptable xml Filtering with Prefix-Caching Suffix Clustering, *Proceedings of the VLDB*, Seoul, Korea,, 559–570.

Cao, Y., Lung, C.-H. and Majumdar, S. (2011). A Peer-to-Peer Model for XML Publish/Subscribe Services, *Proceedings of the Conference on Communication Networks and Services Research*, Ottawa, Canada, 26–32.

Carzaniga, A., Rosenblum, D.S., and Wolf, A.L. (2001). Design and evaluation of a wide area event notification service, *ACM Transactions Computer Systems*, 19(3), 332–383.

Chand, R. and Felber, P. (2008). Scalable distribution of xml content with xnet, *IEEE Trans. Parallel Distributed Systems*, 19 , 447–461.

Chen, S., Li, H.-G., Tatemura, J., Hsiung, W.-P., Agrawal,D., and Candan, K. (2008). Scalable Filtering of Multiple Generalized-Tree Pattern Queries over XML Streams, *IEEE Transactions on Knowledge and Data Engi*neering, 20(12), 1627–1640,.

Dai, L., Lung, C.-H. and Majumdar, S. (2010). Bfilter – a XML message filtering and matching approach in publish/subscribe systems, *Proceedings of the IEEE GLOBECOM*, Miami, Florida, USA, 1–6.

Diao, Y., Altinel, M., Franklin, M.J., Zhang, H., and Fischer, P. (2003a). Path sharing and predicate evaluation for high-performance XML filtering, *ACM Transactions. Database Syst*ems, 28(4), 467– 516.

Diao, Y. and Franklin, M. J. (2003b). Query processing for high-volume XML message brokering, *Proceedings of the VLDB*, Berlin, Germany, 261–272.

Google Cloud Platform (2016). Retrieved March 28, 2016, from https://cloud.google.com/compute/.

Li, M., Ye, F., Kim, M., Chen, H., and Lei, H. (2011). BlueDove: A Scalable and Elastic Publish/Subscribe Service, *Proceedings of IEEE Parallel and Distributed Processing Symp*osium, Anchorage, Alaska, USA, 1254–1265.

Lublinsky, B. (2012). Implementing Pub/Sub Based on AWS Technologies, Retrieved March 28, 2016, from http://www.infoq.com/articles/AmazonPubSub.

Lung, C.-H., Sanaullah, M., Cao, Y., and Majumdar S. (2014). Design and Performance Evaluation of Cloud-Based XML Publish/Subscribe Services, *Proceedings of the 11th IEEE International Conference on Services Computing*, Anchorage, Alaska, USA, 583-589.

Ma, X., Wang, Y., Qiu, Q., Sun, W., and Pei, X. (2014). Scalable and Elastic Event Matching for attribute-based Publish/Subscribe Systems, *Future Generation Computer Systems*, 36(0), 102–119.

NIST, Cloud Computing Synopsis and Recommendations. (2012). Retrieved March 28, 2016, from http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-146.pdf.

Tian, F., Reinwald, B., Pirahesh, H., Mayr, T., and Myllymaki, J. (2004). Implementing A Scalable XML Publish/Subscribe System Using Relational Database Systems, *Proceedings of SIGMOD*, Paris, France, 479–490.

Troan, O., Miles, D., Matsushima, S., Okimoto, T., and Wing, D. (2014): IPv6 Multihoming without Network Address Translation, IETF RFC 7157, March, 2014.

Saxena, P. & Kamal, R. (2013). System architecture and effect of depth of query on XML document filtering using PFilter, *Proceeding of the 6th International Conference on Contemporary Computing (IC3)*, 192–195.

Sun, W., Qin, Y., Yu, P., Zhang, Z., & He, Z. (2008). HFilter: Hybrid Finite Automaton Based Stream Filtering for Deep and Recursive XML Data, *Proceedings of the 19th International Conference of Database and Expert Systems Applications*, Turin, Italy, 566–580.

Authors

Chung-Horng Lung received the B.S. degree in Computer Science and Engineering from Chung-Yuan Christian University, Taiwan and the M.S. and Ph.D. degrees in Computer Science and Engineering from Arizona State University. He was with Nortel Networks from 1995 to 2001. In September 2001, he joined the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he is now an associate professor. His research interests include: Communication Networks, and Software Engineering, and Distributed Systems.

Yang Cao received her Ph.D. in Computer Science from Carleton University in 2012. Her research interests include XML and XPath techniques, distributed systems, cloud deployment, performance analysis of distributed software systems, and software engineering methodologies. She is now a post-doctoral fellow at the University of Illinois at Urbana–Champaign.

Mohammed Sanaullah received his Masters degree in Electrical and Computer Engineering from Carleton University in 2013. His current areas of interest are cloud computing, networking and network virtualization technologies. He is currently working as a Network Operations Analyst in Ottawa.

Shikharesh Majumdar is a Professor and the Director of the Real Time and Distributed Systems Research Centre at the Department of Systems and Computer Engineering at Carleton University in Ottawa, Canada. He represents Carleton University in the board managing the *Huawei-TELUS Centre of Innovation for Enterprise Cloud Services* that focuses on collaborative research among Huawei, TELUS and Carleton. Dr. Majumdar is a member of the faculty team actively involved with Carleton University's *Canada-India Centre for Excellence*. He holds a Ph.D. degree in Computational Science from University of Saskatchewan, Saskatoon, Canada. His research interests are in the areas of cloud and grid computing, operating systems, middleware and performance evaluation. Dr. Majumdar actively collaborates with the industrial sector and has performed his sabbatical research at Nortel and Cistech. He is the area editor for the Simulation Modelling Practice and Theory journal published by Elsevier. Dr. Majumdar is a member of ACM and IEEE and was a Distinguished Visitor for the IEEE Computer Society from 1998 to 2001.