On the Performance of Two Topologies for a Distributed Broker Hierarchy of Publish-Subscribe Middleware

Dilshan Amadoru[1], Hasini Gunasinghe[1], Chamini Hasanga[1], Prabath Abeysekara[1], Vishaka Nanayakkara[1] and Srinath Perera[2]

[1]Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka.

[2]WSO2 Inc., Sri Lanka.

Abstract— **Distributing the right information at the right time is an apparent need of the fast moving globe. Publish-subscribe communication paradigm plays a major role in the systems built to accomplish this need. It has been adopted in variety of today's business domains such as mobile communication, database integration, road traffic visualization etc. With large-scaled, distributed and heterogeneous nature of these systems, there is a high demand for an efficient, scalable and interoperable messaging middleware which is capable of handling the increasing load of messages. As discussed in earlier works, network of messaging brokers that collaboratively act as a single entity provides a scalable architecture for such publish-subscribe middleware. This paper describes two algorithms named tree and cluster that we designed which are used as two approaches for topology of the broker network and evaluates the performance of the distributed publish-subscribe middleware with respect to those two topologies.**

I. Introduction

Thousands of entities around the globe exchange massive amount of information in every second. Distributing the right information on right time at the right place is an essential need that the technology tries to fulfill through the fast growth of internet. Distributed and large scaled nature of the systems has resulted in large number of interactions between heterogeneous entities. When publishers of information and the people who are interested in receiving that information are distributed all over the world, communication becomes inefficient and non-scalable if a traditional tightly coupled, synchronized and P2P type of communication mechanism is used. Therefore many distributed applications have given increased attention to publish-subscribe communication paradigm [2] that has favorable features such as loose coupling and scalability in small installation. Publish-subscribe communication model offers full decoupling between communicating entities in three dimensions named space, time and synchronization as described by Patrick Th. Eugster et al. in "The Many Faces of Publish Subscribe" [2].

Some of the practical use cases of publish-subscribe systems are news publication, database integration as in Oracle [11], distributed streaming systems as in NaradaBrokering [13] and message oriented middleware as in Tibco Rendezvous [14]. In the case of a news publication system, there are a large number of news publishers around the globe and subscribers who subscribe to different news topics. If these news publishers and subscribers interact in P2P manner, the system will not be able to handle the growing number of interactions. A messaging broker who provides mediation between publishers and subscribers can be deployed as the messaging middleware which facilitates publish-subscribe communication model, to solve the above problem. Such a broker accepts subscriptions from subscribers on behalf of publishers and stores them with it. When a publication arrives, broker filters out the matching subscriptions of interested subscribers and delivers the publication to them.

Nevertheless, such a middleware being developed as a single module has not been the ultimate solution. As the message traffic becomes intense, a single module can fail since it has a limitation of processing power. This is the core issue that Wihidum--our research prototype, which is composed of a hierarchical network of broker nodes and which facilitates publish-subscribe communication model in the context of web services in accordance with WS-Eventing specification [1]--addresses as a distributed middleware. The main challenge was to come up with proper mechanisms of arranging the brokers according to a particular topology and defining communication channels among them in such a way that the distributed middleware of broker network as a whole can provision increasing number of messages. We observed that the hierarchy of broker nodes shows different characteristics in performance wise when arranged according to two different topologies.

The main contributions of this paper are as follows:

- We designed the two algorithms named "Tree" and "Cluster" which were implemented to organize the network of broker nodes in a hierarchical manner (Section V). They define two different routing mechanisms among broker nodes. When designing the two algorithms, we have taken favorable features from existing algorithms such as the work carried out by Shrideep Pallickara et al. in Narada brokering [4], Dwi H. Widyantoro et al. in "An Incremental Approach to Building a Cluster Hierarchy" [6], ActiveMq [7] and IBM WebSphere MQ [9].
- We compare and contrast the aforementioned two approaches in terms of their performance with respect to throughput and latency with the support of measures obtained from performance testing.

This research paper is structured as follows. Section II describes the related work in the area of publish-subscribe model. Section III and IV together combine the solutions presented by Wihidum. Section V describes the performance evaluation carried out on Wihidum and discusses the characteristic of Wihidum in performance wise with respect to two topologies. Section VI concludes the paper eventually.

II. RELATED WORK

Multiple publish /subscribe middleware has been proposed in literature. Among them, NaradaBrokering (NB) [13] is an event brokering system designed to run on a large network of co-operating broker nodes. It supports management of subscriptions based on SQL, Regular Expressions and XPath queries. NB incorporates an efficient, reliable and failure resilient routing and matching algorithm based on a cluster topology [4] that guarantees the delivery of information from producers to consumers.

Oracle Advanced Queuing provides database integrated message management functionality and asynchronous communication on top of oracle database using Oracle Net Services. They support both point to point and publish-subscribe messaging facilities. Messages can be stored persistently, propagated between queues on different machines and databases, and transmitted [11]. Due to the support of the strong database backend, this comes with favorable features such as persistency and reliability. But it needs the usage of gateways when communicating with other messaging middleware. In contrast, Wihidum uses minimal number of database operations and it is interoperable by itself.

TIBCO Rendezvous is another messaging solution for data distribution applications due to its low latency and real-time high throughput [14]. TIBCO Rendezvous has optimized a fully distributed daemon-based peer-to-peer architecture which eliminates bottlenecks and single points of failure. Messages in TIBCO Rendezvous are self-describing and platform-independent [15]. This feature is also achieved in Wihidum because of the interoperability of web services.

IBM WebSphere MQ [10] is another messaging backbone which facilitates the information flow across distributed entities. While delivering information either synchronously or asynchronously depending on the needs of the application, it can also reduce the risk of data being lost when applications, web services or networks fail. IBM WebSphere Message Broker [9] is a fast and flexible application integration tool, which can work closely with the WebSphere MQ to deliver a comprehensive publish-subscribe facility.

Apache Savan project [8] enables publish-subscribe communication model for web services hosted in Axis2. It implements the WS-Eventing specification and it is developed as a module on top of Axis2. As a publish-subscribe tool it provides subscription management, filtering and persistence with an embedded sqlite database [3]. Being a single module, Savan does not address performance scalability on which Wihidum focuses in its solution.

III. INTRODUCTION TO WIHIDUM

Wihidum is a distributed publish-subscribe middleware which is composed of a hierarchy of broker nodes. A single broker node is a self contained entity in which the overall functionality of a messaging broker (Section I) is logically encapsulated. A broker node is implemented as a web service [5] in accordance with WS-Eventing [1]. Hence, Wihidum provides interoperability for the heterogeneous systems where the clients of publish-subscribe systems can come from.

The broker hierarchy acts as a single unit and enables users to publish or subscribe using any node in the hierarchy, and just like with one broker unit, the broker hierarchy makes sure that all subscribers get notified of events according to their subscriptions regardless of from which node events have been published or from which node subscribers have subscribed. The broker hierarchy is formed by organizing any number of broker nodes in a particular topology. Current Wihidum implementation comes with two topologies and the flexibility to switch between the two topologies based on the requirements of the application domain. Detailed description of algorithmic solutions of Tree and Cluster topologies are included in the Section IV.

IV. TOPOLOGIES

In a distributed broker network, the topology defines how the brokers are connected, how communication between them happens, and how the subscriptions and publications are routed in order to achieve efficiency in the entire middleware. Wihidum implements two topologies "Cluster" and "Tree".

A. Cluster Topology

Cluster topology, organizes the broker network in a hierarchical structure with three levels in the hierarchy, each level having a unit controller. The main reference we used to define the organization of clustered broker network is NaradaBrokering project [13] whose clustering algorithm is defined in [4].

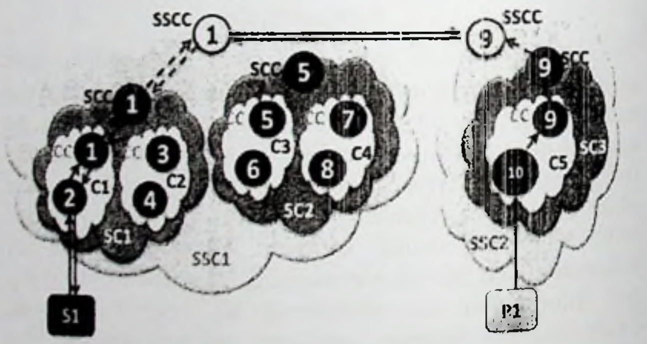


Figure 1: Sample clustered broker network

Organization: Figure 1 illustrates how ten broker nodes are arranged in a cluster topology. Three levels of cluster units in the hierarchy are named as cluster (C), super cluster (SC) and super-super cluster (SSC). Each cluster unit has a unit controller, named by appending the term "controller" to the unit name such as cluster controller (CC), super cluster controller (SCC) and super super cluster controller (SSCC). As in the figure 1, broker nodes 1 and 2 have formed the cluster unit C1. And the cluster units C1 and C2 form the super cluster unit SC1. Again such two super cluster units SC1 and SC2 form the top most cluster unit in the hierarchy-SSC1. The above clustered broker network is consisted of two super-super cluster units.

Communication: In this topology, each broker communicates only with the cluster controller of the cluster unit it belongs to; minimizing the number of communication channels. This is in contrast to NaradaBrokering routing algorithm [4] where there exist multiple channels among the brokers within the same cluster, mainly to achieve failure resiliency. Unit controllers communicate with their super, sub and peer (only in the case of SSCC) unit controllers and with the broker nodes of their cluster units. In that way, unit controllers act as a gateway to the outside of a cluster unit in the network. How subscriptions and publications are propagated upward and downward the hierarchy through the defined communication channels, is shown in figure 1 and explained below.

Subscription management and delivery of publications is accomplished by the inter broker communication and routing mechanism. When a subscription is received at a broker node, it is persisted there as came from the subscriber and is forwarded to its super unit as a broker-subscription with the topic of the original subscription. The super unit persists the subscription and forwards to its super unit and the chain goes on until it is received at a SSCC unit. The SSCC unit then forwards it to its peer units which are also SSCC units. At that point the propagation of the subscription terminates. In the other scenario, where a broker node has already subscribed to its super unit the propagation of the subscription does not happen beyond that node and that criterion minimizes the messages passed between the broker nodes. When a publication is received at a broker node on a certain topic, the subscriptions and broker subscriptions saved with that particular broker node are filtered on the topic and the publication is delivered to the corresponding subscribers and broker-subscribers respectively. Then the publication is propagated along the path of super units just as with subscription routing as described above except that publication will not be forwarded to the peer SSCC units when it is reached at a particular SSCC node unless the peer SSCC units have already subscribed on that topic.

Failure recovery: If a primary unit controller fails, we appoint secondary unit controller for all the sub units inside one unit to prevent the communication path being disturbed. Above type of a failover mechanism is used in Apache ActiveMQ [7] project. Secondly, to prevent the loss of stored subscriptions if any broker node fails, replication of database and restoring it with a new node is proposed.

Advantages: Cluster topology avoids single point of failure unlike in tree topology which has only one root node which gets the entire load of the tree. It also balances the load at each level of clustering in the hierarchy. There is also a mechanism to recover the failure of a unit controller. Although the hierarchy is not simple as in tree topology which is described in section IV B, this topology is more desirable in a large scale distributed middleware.

B. *Tree Topology*

The Tree topology which organizes the brokers in to a logical tree like structure is easy to implement and manage. It uses minimum number of communication channels between brokers.

Organization: The main reference for defining the tree topology is the IBM WebSphere MQ [10] broker network. Figure 2 shows an example of eight brokers laid out according to tree topology.

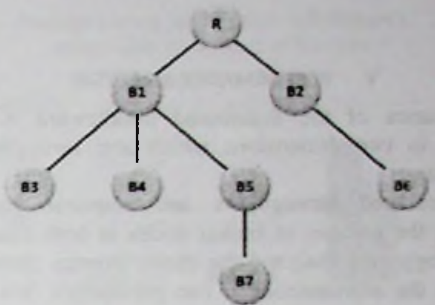


Figure 2: Example tree topology network

Communication: In tree topology each broker communicates only with its parent and children, minimizing the number of communication channels. A broker consolidates all the subscriptions that are registered with it, whether from subscribers directly or from other brokers.

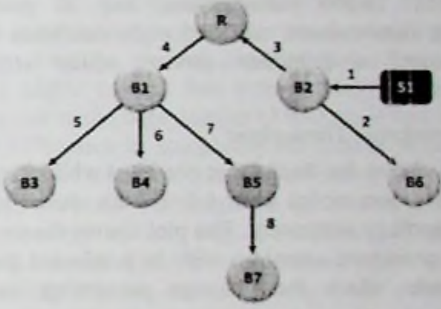


Figure 3: Subscriptions routing in tree topology

Note that the arrow heads in Figure 3 indicates the direction of message propagation and the numbers indicate the steps involved in order. If a subscription already exists in one of the broker nodes it won't propagate the message in that path thereafter.

When an application publishes information, the broker who receives the publication, forwards it to all its neighbor nodes and external subscribers which have valid subscriptions for it.

Advantages: The main advantage of the tree topology is its simplicity in structure unlike the more complicated cluster topology. This topology is preferred with small scale applications which prefer ease of management over high performance.

C. *Load Balancing (for both tree and cluster)*

The broker hierarchy needs a load balancer for two purposes. First, for the broker hierarchy to be viewed as one single messaging broker form outside, there should be a globally available common reference point to which the users can connect. Second, there should be a proper mechanism to distribute the requests coming to the middleware, uniformly across all broker nodes. In order to fulfill those two purposes Wihidum has an admin module which implements a simple load balancing algorithm similar to DNS round robbing [12]. Publishers and subscribers first send a request to it and receive a subset of addresses of existing broker nodes to which the events and subscriptions should be sent afterwards.

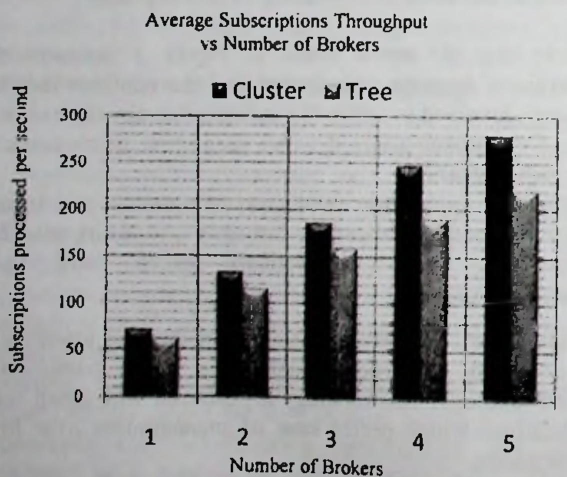
V. PERFORMANCE ANALYSIS

Performance of the distributed middleware is mainly evaluated in two dimensions which are: *throughput* and *response time.*

We measured throughputs and response times by increasing the number of broker nodes in both cluster and tree topology and analyzed the characteristics shown with respect to the aforementioned two parameters. We carried out a performance testing in the Advanced Computing Lab of Department of Computer Science and Engineering, University of Moratuwa. Each computer had Pentium(R) 4 processor, 2.8 G Hz, having 512 MB and running Windows XP. In order to provide sufficient CPU workload in the computers which hosted the broker services we used number of computers hosting subscriber services, listener services and publisher services. To provide workload for one broker service, in subscription throughput measurement, we used eight machines each of which hosted a subscriber service which sent 10,000 subscriptions and in publication throughput measurement we used eight machines each of which hosted a publisher service which sent 2000 publications.

A. Subscription Throughput

Graph 1 shows the throughput observed while varying the number of broker nodes from 1-5 in both cluster topology and tree topology separately. The plot shows the number of messages processed overtime with in a selected period of 100 seconds where the message processing has been stabilized.



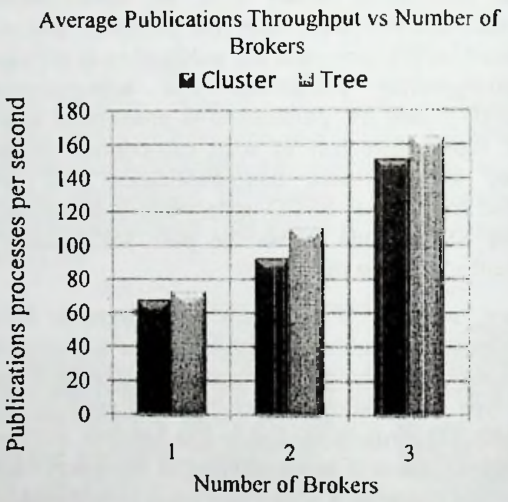
Graph 1: Variation of subscriptions throughput with the number of broker nodes in the middleware with regard to cluster topology and tree topology

From graph 1 we can witness that both the topologies gain the ability of handling more subscription requests as the increment of the deployed broker nodes which is due to utilizing increasing number of processors in distributed manner. According to the routing algorithms of both the topologies, if a particular broker has received a subscription or a broker-subscription on a certain topic before, it has already established its routing paths for that topic and hence there will be no routing cost involved on future subscriptions on the same topic. This criterion also helps to increase the subscription throughput of two topologies in the long run.

In further analysis we can observe that broker network arranged in cluster topology shows a higher throughput than that of tree topology. This is due to the two different basic routing mechanisms used in two topologies. In the case of cluster topology, a subscription is routed only along its super units which can be at most 4 nodes along the path even in a complete network of three levels of clustering, where as in tree topology, there is a high possibility that a newly arrived subscription is routed to all the nodes in the broker network along the neighbor nodes. We can also see that throughput does not get doubled when going from one broker to two brokers because the message routing overhead is involved as the number brokers are increased.

B. Publication Throughput

Practically, the number of publication messages received at a publish-subscribe middleware like Wihidum, will outnumber the subscription messages. Hence, publications throughput is of major importance, when considering the performance of Wihidum.



Graph 2: Publication throughput variation with number of broker nodes in middleware under cluster and tree topologies

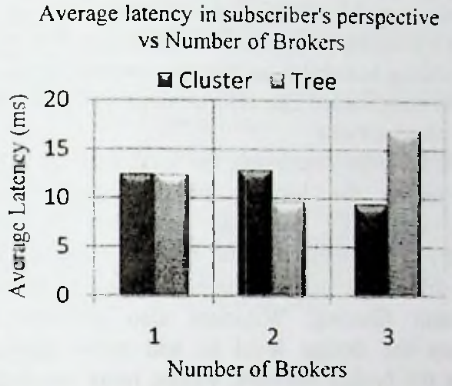
Graph 2 shows the publications throughput observed while varying the number of broker nodes from 1-3 in both cluster topology and tree topology separately. The plot shows the number of publications processed overtime with in a selected period of 100 seconds where the message processing has been stabilized. The publication throughput increases at an average rate of 70% for both the topologies when a new broker is added to the middleware.

Unlike the subscription throughput, the publication throughput is higher in the tree topology compared to the cluster topology. This is due to the algorithmic designs of the two topologies as described in Section IV. In the tree topology the routing of publications only happens in paths where there are interested subscriptions exist. But in the cluster topology irrespective of the available subscriptions, all the publications are routed upwards in the hierarchy, only through the controller nodes up to the SSCC units. In other words, for a particular publication arrived at a broker node in a clustered broker network which has all three levels of clustering, there will be a maximum of three broker-to-broker message routing channels across the controller nodes of the broker hierarchy in addition to the delivery of publication along the existing subscribed paths.

C. Response Time

Response time of Subscription: Subscriptions response time or in other words: latency is measured in two perspectives named latency in subscriber's perspective and latency in broker network's perspective.

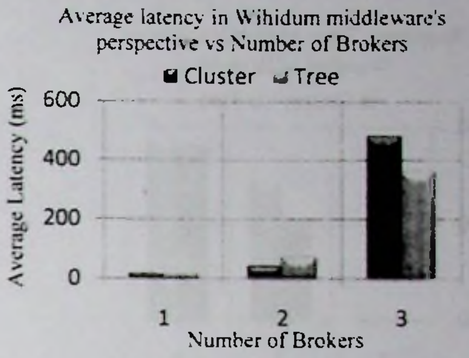
Latency in subscriber's perspective is measured as the average time between arrival of a subscription at a broker node and sending of subscription response to the particular subscriber. Following graph shows the measured latency in subscriber's perspective with broker networks up to three brokers in the hierarchy with respect to both the topologies.



Graph 3: Variation of average latency in subscriber's perspective with the number of broker nodes in two topologies

From the graph, we can observe that irrespective of the growth of the number of broker nodes, both topologies present latency values which lay in a small range, between 9-17 ms, to subscribers. This is achieved through the asynchronous routing mechanism used in the middleware. When a subscription is arrived at a broker node, it is persisted and routing is handed over to an asynchronous process and the response is sent to the subscriber. In this way, routing latency (see Graph 4) is not involved in the latency that middleware presents to the subscribers. With regard to this performance measure, there is no difference between the two topologies; since the same sequence of steps is performed until a subscription response is sent irrespective of the underlying routing algorithm. This is a favorable feature of Wihidum which makes it suitable to be used as a large scale distributed middleware that presents least latency to the users irrespective of the number of broker nodes in the network.

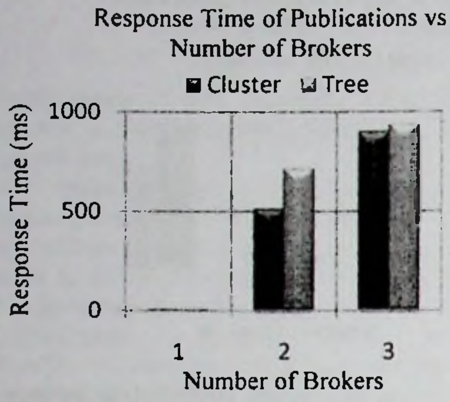
Latency in broker network's perspective is measured as the average time between arrival of a subscription at a broker node and its completion of processing, at the top most broker in the hierarchy in the case of cluster topology and at the last broker node in the routing path in tree topology. Following graph shows the measured latency in Wihidum's perspective with broker networks up to three brokers in both topologies.



Graph 4: Variation of average subscription latency in Wihidum middleware's perspective with the number of broker nodes with regard to two topologies

The measured latency values shown in the graph includes the routing cost and the cost of processing the subscription and broker-subscriptions at each node along the routing path. Since the routing cost is affected by the prevailing network conditions, latency values can have a large range of variation. Further, we can observe that cluster topology shows a higher latency than tree topology with 3-brokers which is due to the organization of brokers in two topologies and the difference in complexity of routing. Since routing cost increases with the number of levels in the broker hierarchy, here we increased the number of levels in the cluster topology by one with the addition of each broker node which resulted in 3 levels in cluster topology where as there are only 2 levels in tree topology with a network of 3 brokers. Routing algorithm of cluster topology is more complex where role of the unit controller (see Section IV) is checked before executing the relevant routing mechanism at each broker in contrast to tree topology where a simple routing mechanism is performed for all the brokers in the topology.

Response time of Publications: Publication process is a one way process at the publishers end. That is, whenever a publisher publishes; it is an event notification where publisher doesn't get a response with respect to his publications. Hence, in a publisher's perspective response time of publications is irrelevant. Nevertheless, architecture of publication process of Wihidum has ensured an uninterrupted publication inflow by enqueuing the publications and by further concurrent processing of them. Therefore, the only response time measure relevant to publication messages are the time taken to entirely finish the processing, routing and delivering a particular publication over the broker network. The publication response time was obtained by averaging the calculated response times for number of publications arrived at the middleware in a certain time period when the publication process is stabilized.



Graph 5: Response times of publications under tree and cluster topologies of Wihidum broker network

Graph 5 shows the publications response time observed while varying the number of broker nodes from 1-3 in both cluster topology and tree topology separately. Response time of publications depends on number of stored subscriptions and the number of brokers in the hierarchy. When the number of subscriptions is higher for a given publication, the response time also increases because of the relatively higher workload. The time to route the publication message adds up with the increasing number of brokers. Hence, in general case, the plot shows a gradually increasing value for the publication response time for a growing broker network under both topologies.

Despite of the higher publications throughput results of tree topology than the cluster topology, we can witness that the cluster topology shows a better response time for publications in graph 5. In the tree topology, publications are always routed across the subscribed paths over the network. In case of cluster topology it can be a combination of simple upward routing in the hierarchy and routing across a subscribed path downwards the hierarchy. Routing across subscribed paths includes the cost of filtering of stored subscriptions based on the topic of the publication, which is not present in the normal upward routing. This difference was visible in the better performance of clustered broker network in graph 5.

Furthermore, we can identify that both the broker topologies have controllable measures to get better publications response times as the network grows by minimizing the number of communication channels. According to the algorithms described in Section IV, tree topology can increase the limit for the number of children of a particular broker node whereas the cluster topology can increase the number of broker units per cluster unit in order to reduce the number of communication channels that a particular message should pass across the network.

VI. CONCLUSION

The In this paper we present two algorithms for two topologies in a distributed broker network. We discussed, compared and contrasted in detail different theoretical and practical aspects of the two topologies.

According to the results we obtained from the performance testing we conducted on Wihidum with respect to two topologies, we could observe that as the number of broker nodes grows, both publication and subscription throughput of the middleware increases significantly. This is achieved by distributing the load among the broker nodes and making them communicate in an efficient manner according to the two routing algorithms. Comparing and analyzing the obtained throughput results and the theoretical aspects of two topologies, we can conclude that tree topology is suitable to be used in medium scale middleware due to its higher publication throughput and cluster topology is suitable to be used in large scale or rapidly growing distributed middleware due to its avoidance of single point of failure.

This application independent middleware can be deployed in variety of domains. Since the broker nodes which compose the middleware are developed as web services, interoperability is provided for large scale systems of which clients can come from heterogeneous platforms. Another interesting thing with this solution is that middleware can be gradually grown larger by adding more broker nodes as the load increases by minimizing resource wastage at the initial stage and providing scalability as the requirement arises. In that aspect this middleware can also be deployed and made available as a cloud service.

Future work includes improving the formation of broker network by taking the locality of the broker in to consideration, improving the load balancing algorithm which currently implements a round robin mechanism and extending the filtering criteria to support advanced features such as content filtering. Wihidum also provides the flexibility from the design level to add more topology algorithms for the broker network. Hence more topologies could be implemented on Wihidum and they could be analyzed and compared in performance wise in the future.

REFERENCES

[1] Davis D., Malhotra A., Warr K. and Chou W. Web Service Eventing (WS-Eventing), September 24, 2009. Retrieved March 02, 2010, from W3C: http://www.w3.org/TR/2009/WD-ws-eventing-20090924/.

[2] Eugster, P.T., Felber, P.A., Guerraoui R. and Kermarrec, A.M. The Many Faces of Publish Subscribe. in *ACM Computing Surveys*, (June 2003), 114-131.

[3] Kumarage D. Web Services Eventing with Apache Savan/C. Retrieved June 5, 2009, from WSO2 Inc.: http://wso2.org/library/3149#Filtering.

[4] Pallickara S. and Fox G. On the Matching of Events in Distributed Brokering Systems. in *International Conference on Information Technology: Coding and Computing (ITCC'04)*, (2004), IEEE Computer Society.

[5] Weerawarana S., Leymann F., Curbera F., Storey T. and Ferguson D.F. *Web Services Platform Architecture*. 2005.

[6] Widyantoro D.H., Ioerger T.R. and Yen J. An Incremental Approach to Building a Cluster Hierarchy. in *IEEE International Conference on Data Mining*, (2002).

[7] Apache ActiveMq, 2004-2008. Retrieved October 2, 2009, from The Apache Software Foundation: http://activemq.apache.org/.

[8] Apache Savan/C. Retrieved May 30, 2009, from The

Apache Software Foundation: http://ws.apache.org/savan/c/.

[9] IBM-WebSphere Message Broker. Retrieved March 25, 2010, from IBM: http://www-01.ibm.com/software/integration/wbimessagebroker/.

[10] IBM-WebSphere MQ. Retrieved March 29, 2010, from IBM: http://www-01.ibm.com/software/integration/wmq/.

[11] Introduction to Oracle Advanced Queuing. Retrieved March 29, 2010, from Oracle Corporation: http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96587/qintro.htm

[12] Simple Failover-DNS Round Robin Load balancing, 2004-2010. Retrieved Feb 27, 2010. from JH Software: http://www.simplefailover.com/scenario3.aspx.

[13] The NaradaBrokering Project @ Indiana University. Retrieved June 30, 2009, from Pervasive Technology Labs at Indiana University: http://www.naradabrokering.org/deployments/index.html.

[14] TIBCO Rendezvous. Retrieved: March 29, 2010, from TIBCO Software Inc.: http://www.tibco.com/software/messaging/rendezvous/default.jsp.

[15] TIBCO Rendezvous datasheet, 2008. Retrieved April 01, 2010, from TIBCO Software Inc.: http://www.tibco.com/multimedia/ds-rendezvous_tcm8-826.pdf.