# PUBLISH/SUBSCRIBE NETWORK INFRASTRUCTURE BASED ON WEB SERVICE NOTIFICATION

**RUISHENG SHI, YANG ZHANG, BO CHENG, JUNLIANG CHEN**
State Key laboratory of Networking and Switching Technology, Beijing University of Posts & Telecommunications, Beijing 100876, China

## ABSTRACT

To accommodate real-time dynamic large scale service composition requirements, Event-Driven Service Oriented Architecture (EDSOA) is introduced to solve the shortcomings of Service Oriented Architecture (SOA). Our EDSOA service execution platform is built on Web Service Notification based distributed topic-based publish/subscribe infrastructure services. This paper presents our innovations at system level on the design of publish/subscribe infrastructure service. Several service systems are developed based on this platform. The platform provides great flexibility and many advanced features for service development and deployment. Through project practices, the platform was proved to simplify the development work of service system significantly.

**Keywords**: *Topic-Based Publish/Subscribe, Distributed Event-Based System, Service Computing, Web Service Notification, Routing*

## 1 INTRODUCTION

With the rapid development of large scale distributed network services and mobile internet computing, the distributed publish/subscribe systems as a well known mode of Event Driven Architecture (EDA) attracted great attentions and in-depth research and application in academia [1-3] and industry [4-6].

Service Oriented Architecture (SOA) has been widely applied in the past decade. SOA is becoming the dominant architecture to build large scale network services [7] and business process [8]. The BPEL (Business Process Execution Language) as Web Service composition standards has been widely deployed in the enterprise information system. It has been the de facto standard for service orchestration.

SOA and EDA as two kinds of important distributed service architecture have been evolved independently in the past years. The most salient difference between EDA and SOA is how to compose service process with business components. SOA is based on passive request/response mode, while EDA exhibits the active event notification. SOA is based on explicit invocation mode, and EDA is based on implicit invocation mode. In implicit invocation mode, web services need to register its concerned event type to multiple event producers. When the event happens, these event producers will inform registered Web service, and trigger the corresponding event handling logic to complete the invocation.

How to take the advantages from these two kinds of architecture has become a hot research topic in recent years. Juric et al [9] proposed to extend BPEL to support the EDA structure. The business component works as event producers and/or event consumers. These components complete service orchestration of business process through interactive cooperation between producers and consumers. BPEL language is the standard to develop composite services. In traditional design, BPEL engine usually employs centralized architecture. Scalability is addressed by replicating the engine. When a business process needs to be scaled to meet heavier processing needs, the BPEL engine’s clustering algorithm automatically distributes processing across multiple engines. Li et al [10-12] introduce the distributed content-based publish/subscribe agency network architecture. A large flow is decomposed as many fine grain activities and the centralized business process are mapped to control flow expressed by publish/subscribe semantics. This approach removes the scalability bottleneck of a centralized orchestration engine and exhibits significant performance improvements.

OASIS (Organization for the Advancement of Structured Information Standards) has issued Web Service Notification (WS-Notification, WSN) specification, which defines a set of standardized communication models of publish/subscribe system based on Web Service [13-16]. However, WSN only defines the basic event notification model. Based on WSN standard, researchers has proposed various extensions and practical design solutions. Quiroz et al proposed notice agency services based on Distributed Hash Table (DHT) thus using DHT to realize subscription management and notification delivery [17]. Labey et al expanded WSN standard to support complex events and ECA (Event Condition Action), and manage the life cycle of ECA rules [18]. Gu et al proposed the design and realization of WSN based on cloud queue model [19].

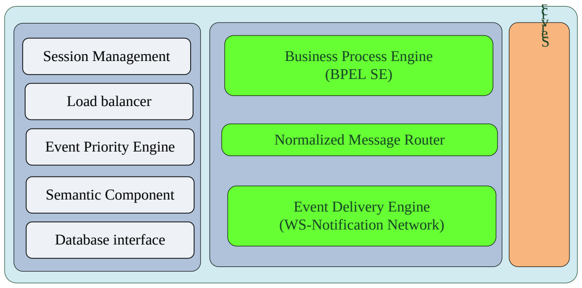

Session Management
Load balancer
Event Priority Engine
Semantic Component
Database interface
Business Process Engine (BPEL SE)
Normalized Message Router
Event Delivery Engine (WS-Notification Network)

*Figure 1: Service Execution Environment*

In the past years, the Service Execution Platform developed by Network Service Infrastructure Research Center of BUPT (Beijing University of Posts and Telecommunications) has completed the evolution from the traditional SOA architecture to EDSOA (Event Driven SOA) architecture.

WSN and JBI build the bridge to Web Service world dominated by SOA. Publish/Subscribe network is the foundation to support large scale distributed EDA. Figure 1 shows Service execution environment, which includes BPEL Service Engine, Normalized Message Router based on JBI (Java Business Integration) specification, publish/subscribe overlay network based on WS-Notification, and a series of infrastructure services such as session management, load balance, event priority determination engine, semantic component, database interface, and so on.

This paper systemically explore the design issues and solutions in-depth on topic-based publish/subscribe network aiming at real-time dynamic service composition environment. This paper presents overlay construction based on cluster structure, broker software stack architecture, fault tolerance design and priority-aware low latency routing scheme. Our best practices incorporate the traditional wisdom on distributed publish/subscribe design with innovative design. We would like to share our best practices of these system level technical innovations to research community in this paper.

## 2 PUBLISH/SUBSCRIBE PARADIGM AND WEB SERVICE NOTIFICATION

In this section, we have a brief introduction on publish/subscribe paradigm and Web Service Notification specifications.

### 2.1 Publish/Subscribe Paradigm

The publish/subscribe system has been widely applied in distributed computing environment. It makes the application program easy to achieve loose coupling system structure. In a word, the system includes three kinds of decoupling of in space, time and control flows [20].

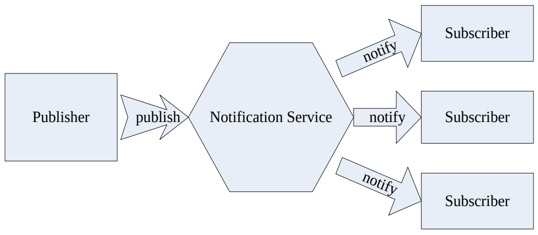


*Figure 2: Decouple in Space*

As shown in the Figure 2, the interacting parties do not need to know each other. The publishers publish events through an event service and the subscribers get these events indirectly through the event service. The publishers do not usually hold references to the subscribers; neither do they know how many of these subscribers are participating in the interaction. Similarly, subscribers do not usually hold references to publishers; neither do they know how many of these publishers are participating in the interaction.

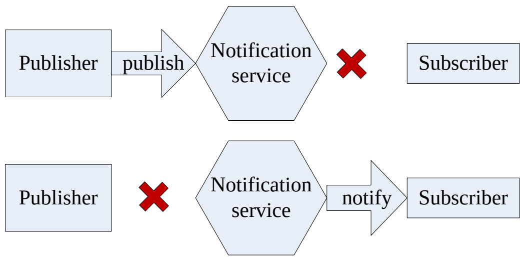


*Figure 3: Decouple in Time*

As shown in the Figure 3, the interacting parties do not need to be actively online at the same time. In particular, the publisher might publish some events while the subscriber is disconnected, and conversely, the subscriber might get notified about the occurrence of some event while the original publisher of the event is disconnected.

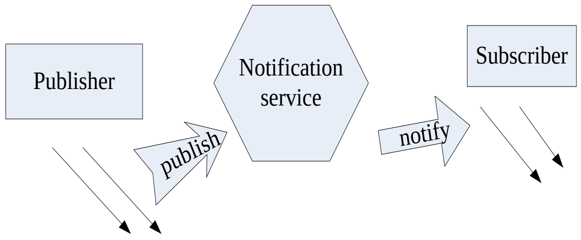


*Figure 4: Decouple in Flow*

Publishers are not blocked and need not waiting while producing events. Subscribers can asynchronously get notifications of the occurrence of an event while performing some concurrent activities as shown in Figure 4. For example, subscribers do not need to "pull" event synchronously. In short, the production and consumption of messages do not happen in the main control flow of the publishers or subscribers.

## 2.2 Web Service Notification

WSN specification includes three standard documents: WS-Base Notification [14], WS-Brokered Notification [15] and WS-Topics [l6].WS-Base Notification standard defines the Web Services interfaces and Web standard message exchanges format for interactions between producers and consumers. This is the base document on which the other WSN specification documents depend. If only point-to-point publish/subscribe notification is required, reading the document and Web service white paper is enough. WS-Brokered Notification describes the operational requirements expected by service providers and requestors that participate in brokered notifications, and defines the Web Services interfaces for notification brokers and the corresponding message format. WS-Topics defines the data format for storage of corresponding theme document, and a mechanism to organize and categorize themes.

WSN standard pointed out a kind of point to point message exchange mode in the WS-Base Notification standard document as shown in Figure 5.

WSN standard pointed out the message forwarding mode based on the broker in the WS-Brokered Notification specification as shown in Figure 6. In this architecture, broker is in charge of interactions between event producers and event consumers.

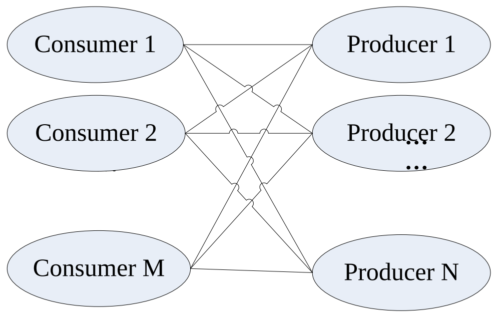


*Figure 5: WS-BaseNotification*

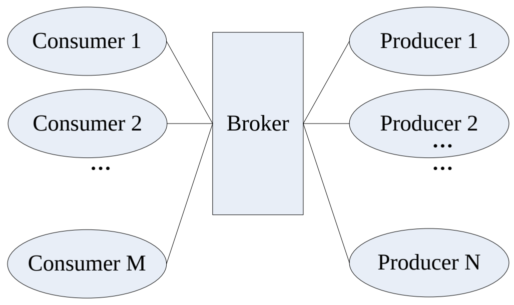


*Figure 6: WS-Brokered Notification*

## 3 SYSTEM ARCHITECTURE

This chapter describes the distributed publish/subscribe system architecture in the service execution platform from several aspects. It includes publish/subscribe model, distributed WSN overlay network scheme, overlay network topology, service infrastructure software architecture.

### 3.1 Publish/Subscribe Model

Publish/subscribe paradigm can be classified into topic-based model, content-based model, type-based model and so on.

Due to the rapid development of large scale geographically distributed network service, topic-based publish/subscribe mode has been recognized and widely used by the academia [2-3, 21] and industry [4-6] as its simplicity and high efficiency.

WSN specification also follows the topic-based publish/subscribe model. The topic-based publish/subscribe system abstractly maps individual topic to distinct communication channels. The topic names contained in the metadata for the system are usually specified as an initialization parameter of the system. As almost all systems provide hierarchical organization model, one significant advantage of topic-based publish/subscribe systems is that the applications can organize topics in accordance with corresponding customized strategies. A subscription to a node in the hierarchy system implicitly subscribes all the subtopics of that node. The topics are divided into more than one topics, subscriber appoints interested sub topics, thus avoiding receiving uninterested events. Each topic is identified by unique name, and provides interfaces for publish and subscribe operations.

## 3.2 Distributed Web Service Notifications

For small scale applications, the centralized broker architecture can be applicable. For large scale applications, system scalability becomes the concerns of many research works. In the distributed broker architecture, the broker is an overlay network composed by a set of broker nodes. Subscription information management and events/notification message routing are completed through the cooperation of broker nodes.

The roles of broker node are classified as access node and routing node as shown in Figure 7. Access node is responsible for client access, which represents the whole overlay network composed of broker nodes for the client and is also known as client home broker node. Routing node is responsible for message routing. A node can act as access node and routing node simultaneously.

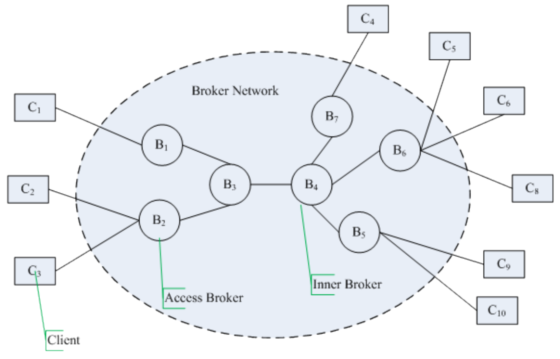


*Figure 7: An Exemplary Publish/Subscribe Overlay Network*

A client can be a publisher and/or a subscriber in the system.

## 3.3 Overlay Network Topology And Cluster Partition

Considering system scalability and management efficiency, broker nodes that participate in publish/subscribe overlay network are organized as two layers structure as shown in Figure 8.

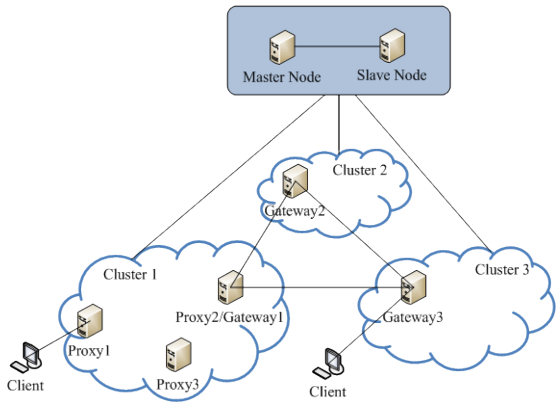


*Figure 8: Overlay Topology of Publish/Subscribe System*

First, brokers in publish/subscribe overlay network shall be divided into clusters. The strategy of cluster partition shall consider geographical distribution, for example, the broker nodes located in the same data center or located in the same Local Area Network (LAN) are preferred to be clustered together. It is not reasonable to make two nodes far away into one cluster. If the cluster is located in the LAN supporting IP multicast, it is possible to exploit the ability of IP multicast to achieve bandwidth-saving. If the cluster does not support IP multicast, the message distribution will be conducted by routing algorithm strategy according to the cluster configuration.

Subscription information management, event distribution and topology management can be managed in two levels separately.

Management node is responsible for topology maintenance and management of system metadata. To ensure the high availability of management node, the master/slave mode is adopted.

Each broker node must belong to some cluster and is assigned an identification which is unique in the cluster. Each cluster has a unique cluster name to distinguish from other clusters. The cluster name and node ID can identify one broker node uniquely. Each cluster must select one broker as the cluster head to communicate with other cluster and management nodes. The cluster works as gateway between clusters.

Each node needs to register to the management node and obtain the certificate authority before joining the overlay network. After that, management node decides which cluster it belongs to according to the request information (such as IP address) and returns the cluster head information to the request node, and the request node joins the cluster by communication with cluster head. At the same time, management node will allocate a backup node for the request node out of cluster.

Routing in overlay network can be classified into intra-cluster routing and inter-clusters routing. Inter-clusters routing is equal to routing between the cluster heads. For routing within a cluster, each cluster can define the message routing strategy in the cluster. This mechanism provides flexibility to achieve the autonomy within cluster.

This hierarchical network architecture effectively reduces the scale of the routing problem and alleviates the workloads of the system management nodes. The system scalability is improved dramatically.

## 3.4 Software Architecture

System consists of the client-side software development kit (SDK) and server-side publish/subscribe infrastructure services as shown in Figure 9.

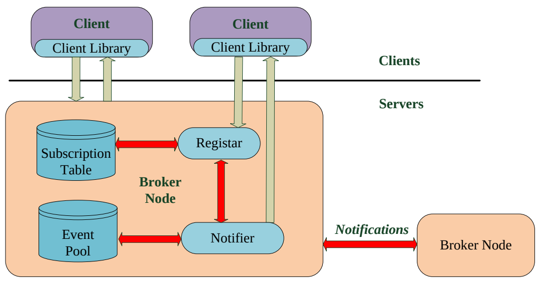


*Figure 9: Software Architecture*

The client SDK provides publish/subscribe interfaces for applications and hides the complexity of the server-side communication protocol. For example, when the original access node fails, the application service request is routed to the new access node through interaction with the management node. This mechanism guarantees uninterrupted service for applications.

The components at server-side include management node and broker nodes. Management node is responsible for client access, cluster partition, node join/leave and topology management. Besides the three basic function of topology management, subscribe management and notification message routing as shown in Figure 10, broker node also need consider many advanced features such as system high availability, service dynamic migration, high utilization of system resources, efficient events delivery mechanism based on data prioritization, and topics aggregation based on semantics.

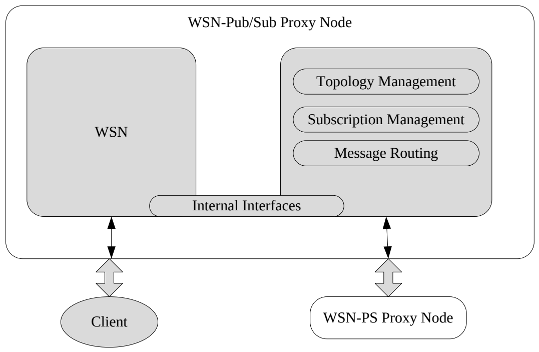


*Figure 10: Function Modules in Proxy Node*

Subscription management module in broker node maintains three flavors of subscription tables: client subscription table named cTable, broker subscription table named bTable, cluster subscription table named gTable. After received the subscription requests from client, broker node update the subscription requests into its cTable firstly. If this subscription has not been registered from other clients, the broker node recognizes this subscription as new subscriptions and broadcast this subscription request within the cluster by its own ID. The broker nodes in the cluster will update bTable after received this broadcast subscription message. Cluster head shall forward the request to other cluster head nodes by its cluster ID if this subscription request has not been registered from other broker nodes in this cluster. All other cluster head nodes will update its gTable while received this intra-cluster subscription requests.

Message routing module in broker node uses the routing algorithm module to calculate Routing table according to the subscribe table. Notification message forwarding module called Notifier in Figure 9 is responsible for forwarding message according to the routing table. If the client is not online, message shall be stored in event pool. The message will be pushed to the client once the client joined the overlay network.

## 4 ANALYSIS ON TECHNICAL ISSUES

This section addressed the core design issues in large scale publish/subscribe infrastructure services from following aspects. The main aspects include high availability and fault tolerance design, cloud-based active push services, routing efficiency with data prioritization, scalable event priority determination engine.

### 4.1 High Availability And Fault Tolerance Design Of Broker Nodes

From the perspective of client applications, the whole publish/subscribe overlay network is a virtual entity i.e. the broker entity in WS-BrokeredNotification specification. For the upper application, subscribe and publish operations are expected to be reliable. The failure of any broke node shall not cause loss of application information (subscription information, event notification message) and interruption of services. The failure of broker node can be divided into several types.

(1) Failure of access node. Within a cluster each access node will be configured with N (N default is 2) backup nodes. The client SDK library can detect failure of access node and guarantee continuous service by negotiations with server side processes. The client SDK will send requests management node to assign one backup node to take over current service. Management node will forward client requests to cluster head node within the cluster and cluster head node selects one access node from backup nodes to manage the current service.

(2) Failure of cluster head. In order to ensure the high availability of cluster head node, when cluster head node fails, the rest of the cluster nodes will choose new representative according to the election algorithm.

(3) Failure of whole cluster. The failure of network or infrastructure issues might result in the whole cluster unavailable. For example, power supply issue, fire disaster, etc. might result in the outage of whole datacenter. The clients of the outage cluster will be taken over by the backup nodes through negotiations with management node according to system configuration.

(4) Failure of backup nodes. Backup nodes are classified into internal cluster backup node and external cluster backup node. When an internal cluster backup node fails, the cluster head is responsible for assigning a new backup node in its cluster to take over the responsibility from the failure backup node. The new backup data can obtain previous data from original node and other backup nodes. When the external cluster backup node fails, the cluster head of the failed node need coordinate with management node find the successor of the failed node. In case of the outage of whole cluster, the management node need reassign the responsibilities of all failed backup nodes to new successors.

The applications are unaware of the message negotiation and routing process, which are completed by the client SDK. This mechanism can greatly simplify the development of applications and guarantee high availability of publish/subscribe infrastructure service to applications.

### 4.2 High Availability Design On Management Node

From section 4.1 we can see that the management node play a critical role in system. The high availability of management node is guaranteed by master/slave backup architecture. When master node is crashed, slave node will take over its responsibilities.

First of all, set a number of redundant backup management node and its priority, and add heartbeat detection function between those management nodes. Once the master management node detection fails, slave management node detects the timeout of master management node heartbeat response message, and becomes master management node after reporting to management system. Then a new redundancy backup is selected from other slave management nodes and synchronizes information backup, waiting for the manual recovery of the original master management node before joining the slave management node queue.

Management node needs to save cluster information (all cluster information) which can be obtained from the root node. When the root node joins, management node will write root node information into files, and if the master management node collapses, then the slave management node is started to read the master node information from the file, recovering the cluster information of slave management node.

Management node updates information including basic configuration update and system topology information recovery, and the update strategies include regular topology information update and synchronous redundancy management node backup.

### 4.3 Integration Of Server Push And Client Pull

If there is no event notification service, the application needs to provide its own notification mechanism. A widely used method is that the client program timely inquires the server and notifies the application when finding changes. This method is simple, but it is clearly not an efficient approach in term of resource utilization since there may be no event happens in most of the time. Furthermore, this approach cannot guarantee notifications to be delivered timely since its timeliness depends on query (polling) interval time. If a large number of clients are using this mechanism, the server will be under great pressure.

The push mechanism can significantly improve the real-time notification, and significantly reduce the server's pressure, thus reducing the waste of resources.

The client may be offline for a long time and can obtain notification message from PullPoint of access nodes when connecting the network again. Because single node's cache resource is limited, so cache resources can be distributed according to time and the importance of cache message. Overdue massage will be deleted from the cache. The massage life cycle in the cache is different according to the importance of message, which is customized by application program. The publish/subscribe network service can provide billing data of cache resource usage to support cloud service model to pay for the resources usage. For example, the resource usage can be measured with storage space multiplied time.

## 4.4 Multiple Prioritized Queues And Scalable Priority Determination Engine

As many types of applications can share one basic service network, various kinds of data often are transferred in one publish/subscribe overlay network. These applications and data have different requirements about timeliness on data delivery. In general, the data can be classified into delay sensitive data and delay insensitive data. Application can be also categorized as (soft) real-time application and non real-time application. Data is useful if and only if it is delivered before deadline [22-24]. This performance aspect can be measured by Events Delivery On-time Rate (EDOR).

Through data prioritization, the employment of multiple prioritized queue technology in broker nodes can significantly improve the measurement of EDOR.

When many applications are executed in one service execution environment, a global priority determination strategy on events is necessary. Otherwise, if an application can set priority identification independently, each application may abuse its priority rights. For example, the non-real-time application may mark its low priority events with high priority label in order to obtain better resources. Obviously, the abuse of priority may result in the failure of date prioritization mechanism.

In practice, there are many challenges for the design of a global Priority Determination Engine (PDE). The PDE is required to provide fast speed, high scalability in terms of the scale of rule database and events throughput metrics. In the distributed environment, how to design a high (events) throughput, low delay priority decision engine, becomes a very challenging problem [25].

Once broker node receives one event message, the broker need label the event with priority flag by interaction with PDE service. The time costs of event priority determination can be divided into two parts. First part is the network communication delay between broker node and PDE service. Second part is the execution time of rule matching algorithm for each event message.

The Bloom filter (BF) data structure is employed to store the summary of priority rule instances and enable fast online query speed with time complexity. The query on BF-based in memory rule database only require k hash calculations on the event instance signature. The online query performance is independent of the number of priority rules. On the other hand, this approach is cache friendly. The query results on signatures can be cached in event priority signature table of broker node. The succeeding event priority query might get result from the event priority signature table in cache memory of broker node as shown in Figure 11. Therefore, most network communication costs are reduced as memory access time is about 100 ns while even in a Data Center the RTT (Round-Trip Time) of the network is about 500,000 ns. The query performance and system throughput are improved significantly [25].

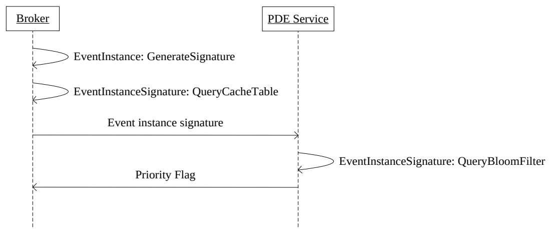

Broker
PDE Service
EventInstance: GenerateSignature
EventInstanceSignature: QueryCacheTable
Event instance signature
EventInstanceSignature: QueryBloomFilter
Priority Flag

*Figure 11: Interaction between Broker Node and PDE Service*

These benefits are at the cost of false positive determination and offline computation efforts. False positive issue is introduced by the employment of Bloom filter. It means that some low priority events may be flagged as high priority label. However, the probability of false positive can be controlled in the acceptable ratio by adjusting the Bloom filter parameters. Moreover, the impacts of false positive issue in practice are trivial and always can be neglected. Second, rule instantiation process might consume large amounts of computing resources and time. Fortunately, this process is offline.

In essence, the false positive rate and complex offline processing are charged for the high performance and scalability of event priority determination engine.

The multiple priority queue scheduling algorithm adopts FCFS (First Come First Serve) in each queue. The scheduling algorithm across multiple queues is designed to ensure that to deal high priority queue as far as possible. Our solution is to assign the time slice (processing weight) according priority. This approach can avoid starvation happened to low priority queue.

## 5 PERFORMANCE EVALUATIONS

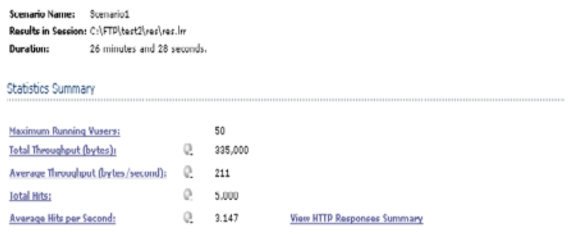
Our performance evaluations have been tested on IBM System x3850 X5 Server with following configuration. We setup 12 virtual machines clustered in six groups at three physical servers. Each virtual machine is configured with 1G memory, 3GHZ CPU, 20G Hard disk and Gigabit network card. The OS image is Windows XP professional.

The loadRunner tool is used to generate the load and evaluate the system performance in term of the hit per second and throughput as shown in Figure 13 and Figure 14.

In this experiment, we configure 50 Vusers to generate 5,000 notifications to one access broker node. The experiment last for 26 minutes and 28 seconds as shown in Figure 12. The total throughput is 335,000 bytes. The average throughput is 211 byte per second.

First, we verify that system functionality from logs in other broker nodes. All messages are delivered to right broker nodes as expected. No errors and no message loss happened.

Figure 13 shows that the number notifications sent to the broker node every second during the experiments. Figure 14 shows that the bytes sent to the broker node every second during the experiments.



**Scenario Name:** Scenario1
**Results in Session:** C:\FTP\test2\res\res.lrr
**Duration:** 26 minutes and 28 seconds.

Statistics Summary

| | |
|---|---|
| Maximum Running Vusers: | 50 |
| Total Throughput (bytes): | 335,000 |
| Average Throughput (bytes/second): | 211 |
| Total Hits: | 5,000 |
| Average Hits per Second: | 3.147 |

View HTTP Responses Summary

*Figure 12: Experiment Summary*

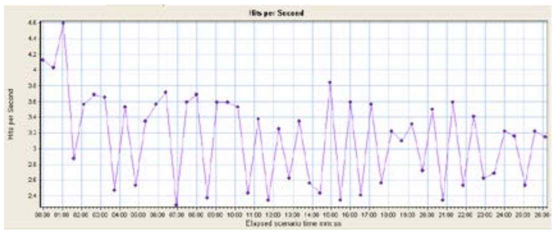


*Figure 13: Hits Per Second In One Broker Node*

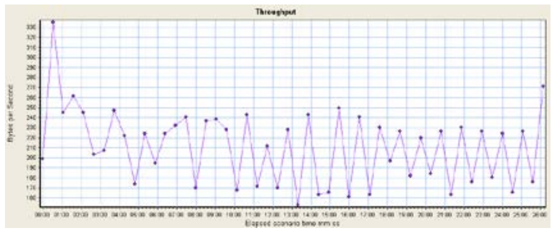


*Figure 14: Throughput Per Second In One Broker Node*

## 6 CONCLUSIONS

Our service execution platform is designed for the evolution from SOA architecture to EDSOA architecture. This paper introduces the design of the publish/subscribe network service infrastructure based on WS-Notification specifications for EDSOA business process service platform, including the system model, the network topology and routing strategy, software architecture. This paper expounds and analyzes the key technical issues in the design, and presents the corresponding solutions.

At present, the framework and the core functions of the prototype system have already been developed. Some advance features are still under development.

The service systems such as emergency management, coal information management and city heating management could be developed based on this platform. Through project practices, the platform was proved to simplify the development work of service system.

## ACKNOWLEDGEMENTS

This work was supported by the National Grand Fundamental Research 973 Program of China under Grant No.2011CB302704; National Natural Science Foundation of China under Grant No.61171102, 61132001, 61001118, 61003067; Project of New Generation Broadband Wireless Network under Grant No.2010ZX03004-001, 2011ZX03002-002-01; Program for New Century Excellent Talents in University (Grant No. NCET-11-0592).

## REFRENCES:

[1] G. Cugola, A. Margara, "Processing flows of information: From data stream to complex event processing", ACM Comput. Surv., Vol. 44, No. 3, pp. 15–84, 2012.

[2] K. Katsaros, G. Xylomenos, and G. Polyzos, "Multicache: An overlay architecture for information-centric networking", Computer Networks, 2010.

[3] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications, Vol. 20, No. 8, pp. 1489–1499, 2002.

[4] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform", Proceedings of the VLDB Endowment, Vol. 1, No. 2, pp. 1277–1288, 2008.

[5] A. Silberstein, J. Chen, D. Lomax, B. McMillan, M.Mortazavi, P. Narayan, R. Ramakrishnan, and R. Sears, "Pnuts in flight: Web-scale data serving at yahoo", IEEE Internet Computing, Vol. 16, No.1, pp.13–23, 2012.

[6] Atul Adya, Gregory Cooper, Daniel Myers, Michael Piatek, "Thialfi: A Client Notification Service for Internet-Scale Applications", Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP), 2011, pp. 129-142.

[7] Jeff Dean, "Evolution and Future Directions of Large-scale Storage and Computation Systems at Google", Proceedings of Symposium on Cloud Computing (SOCC) keynote, June, 2010.

[8] D. Jordan, J. Evdemon, Web Services Business Process Execution Language Version 2.0, OASIS Standard, 2007.http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

[9] M. Juric, "Wsdl and bpel extensions for event driven architecture", Information and Software Technology, Vol. 52, No. 10, pp. 1023–1043, 2010.

[10] Guoli Li, Vinod Muthusamy, Hans-Arno Jacobsen, "A distributed service-oriented architecture for business process execution", ACM Transactions on the Web, Vol.4, No. 1, 2010.

[11] Guoli Li, Vinod Muthusamy, Hans-Arno Jacobsen, Serge Mankovski: Decentralized Execution of Event-Driven Scientific Workflows, SCW 2006, pp. 73-82

[12] Guoli Li, Hans-Arno Jacobsen: Composite Subscriptions in Content-Based Publish/Subscribe Systems. Middleware 2005: 249-269

[13] OASIS Technical Committee. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[14] OASIS Technical Committee. Web services base notification1.3–http://docs.oasis-open.org/wsn/wsn-ws _base_notification -1.3-spec-os.pdf, 2006.

[15] OASIS Technical Committee. Web services brokered notification 1.3 http://docs.oasis-open.org/wsn/wsn-ws _brokered _notification- 1.3-spec-os.pdf, 2006.

[16] OASIS Technical Committee. Web services topics 1.3 – http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf, 2006.

[17] A. Quiroz and M. Parashar, "Design and implementation of a distributed content-based notification broker for ws-notification," Proceedings of 7th IEEE/ACM International Conference on Grid Computing, 2006, pp. 207–214.

[18] S. De Labey and E. Steegmans, "Extending ws-notification with an expressive event notification broker," in Web Services, 2008. ICWS'08. IEEE International Conference on. IEEE, 2008, pp. 312–319.

[19] Gu, P. and Shang, Y. and Chen, J. and Deng, M. and Lin, B. and Li, C. "ECB: Enterprise Cloud Bus Based on WS-Notification and Cloud Queue Model", IEEE World Congress on Services (SERVICES), 2011, pp. 240-246.

[20] P.T. Eugster, P.A. Felber et al., "The Many Faces of Publish/Subscribe, ACM Computing Surveys", Vol. 35, No.2, 2003, pp.114-131.

[21] V. Ramasubramanian, R. Peterson, E. G. Sirer, "Corona: A High Performance Publish-Subscribe System for the World Wide Web", Proceedings of NSDI'2006, pp.73-82, 2006.

[22] Zats, D., T. Das, et al., "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks", SIGCOMM 2012, ACM.

[23] B. Vamanan, J. Hasan, et al., "Deadline-aware datacenter tcp (D2TCP)", SIGCOMM 2012, ACM.

[24] C. Wilson, H. Ballani, et al., "Better never than late: Meeting deadlines in datacenter networks", SIGCOMM 2011, ACM.

[25] R. Shi, Y. Zhang, B. Cheng, J. Chen, "Summary Instance: Scalable Event Priority Determination Engine for Large Scale Distributed Event-based System," Proceedings of IEEE 9th International Conference on Service Computing (SCC 2012), Honolulu, HI, USA 2012.