Poster: A Cloud-enabled Coordination Service for Internet-scale OMG DDS Applications

Kyoungho An and Aniruddha Gokhale
Dept of EECS, Vanderbilt University
Nashville, TN 37212, USA
{kyoungho.an, a.gokhale}@vanderbilt.edu

ABSTRACT

The OMG Data Distribution Service (DDS), which is a standard specification for data-centric publish/subscribe communications, has shown promise for use in internet of things (IoT) applications because of its loosely coupled and scalable nature, and support for multiple QoS properties, such as reliable and real-time message delivery in dynamic environments. However, the current OMG DDS specification does not define coordination and discovery services for DDS message brokers, which are used in wide area network deployments of DDS. This paper describes preliminary research on a cloud-enabled coordination service for DDS message brokers, *PubSubCoord*, to overcome these limitations. Our approach provides a novel solution that brings together (a) ZooKeeper, which is used for the distributed coordination logic between message brokers, (b) DDS Routing Service, which is used to bridge DDS endpoints connected to different networks, and (c) BlueDove, which is used to provide a single-hop message delivery between brokers. Our design can support publishers and subscribers that dynamically join and leave their subnetworks.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications

Keywords

Data Distribution Service, Cloud Computing, Publish/Subscribe, Middleware, Discovery, Coordination

1. INTRODUCTION

Emerging paradigms, such as the internet of things (IoT), connect machines and devices in a loosely couple manner to form intelligent and large-scale systems. The publish/-subscribe (pub/sub) communication paradigm is attractive for these emerging domains since it provides a scalable and decoupled data delivery mechanism between communication peers. However, these new systems need to be scalable, interoperable, and dependable while operating in dynamic environments with entities that may be mobile. Specifically, IoT applications require scalable pub/sub that satisfies a number of quality of service (QoS) properties. The OMG Data Distribution Service (DDS) [5] is a standard specification defining a data-centric pub/sub middleware with many QoS policies, which holds substantial promise for IoT applications.

Despite the many features of DDS, there are multiple challenges in using DDS for internet-scale applications. First, DDS uses multicast as a default transport to discover endpoints (*i.e.*, publishers and subscribers) in a system. If endpoints are located in isolated networks not supporting multicast, these endpoints cannot be discovered by other peers. In addition, because of network firewalls and network address translation (NAT), even if endpoints are discovered, peers may not deliver messages to the destination endpoints. Some DDS broker solutions [6] [2] exist to resolve these issues. Yet, for internet-scale DDS applications where a number of heterogeneous devices and networks exist, a middleware solution to efficiently and scalably discover and coordinate DDS brokers located in isolated networks remains an unresolved issue.

To fill this gap, this paper presents preliminary ideas on PubSubCoord, which is a cloud-based coordination service for geographically dispersed DDS brokers to transparently connect endpoints managed by different brokers to realize internet-scale, pub/sub applications. Our solution extends a pub/sub system architecture from the BlueDove system [4] for scalability and low-latency by leveraging its single-hop message delivery capability between edge brokers (brokers adopting and managing endpoints in a network). The cloud-based approach enables elasticity to the solution.

It is possible that a broker can directly communicate with other brokers without an intermediate layer of brokers, but in such a solution, each broker will need to manage connections to other brokers and deliver a number of messages as well. To resolve this scalability issue, edge brokers can be used as intermediate brokers for other brokers by constructing an overlay network. However, if an edge broker fails, it interrupts data delivery of other brokers because the failed broker is used as an intermediate broker.

Our solution solves both scalability and reliability problems. The inter-edge broker data delivery is managed by a routing broker (intermediate brokers connecting edge brokers), which reduces the number of connections incident on edge brokers and ensures that a failure of edge brokers does

DEBS'14, May 26-29, 2014, MUMBAI, India.
Copyright 2014 ACM 978-1-4503-2737-4/14/05 ...$15.00.
http://dx.doi.org/10.1145/2611286.2611323.

not affect data delivery of other edge brokers. Our approach can also reduce end-to-end latency due to the single-hop data delivery between edge brokers compared to the overlay network of connected edge brokers, which incurs multi-hop data delivery.

In our approach, if a routing broker gets overloaded or fails, the overall quality of service (latency or throughput) provided through the routing broker can be degraded or possibly the service can become unavailable. To overcome this problem, routing brokers are placed in cloud data centers, clustered with others, which elect a leader among themselves. A leader of routing brokers is responsible for load balancing, scaling, and fault-tolerance. If one of routing brokers in a cluster fails, the leader assigns topics adopted by the failed routing broker to avoid service cessation.

Our pub/sub system provides an automatic discovery mechanism between brokers without static configurations of locators (IP address and port) by exploiting ZooKeeper [3], which is a service for distributed process coordination. Brokers connect to ZooKeeper servers as clients to store discovery event information and notify other brokers if brokers are matched by topics.

For mobile publishers and subscribers that are common in IoT, the endpoints spawned in devices can be reassigned to another edge broker as a range of a served network domain possibly changes. For this scenario of endpoint mobility, routing decisions of brokers need be made in a reliable and timely manner. In our approach, notifications between brokers can be done by utilizing ZooKeeper because of its ease, scalability, and reliability in terms of data delivery and data consistency.

The remainder of this paper is organized as follows: Section 2 provides background information on the underlying technologies; Section 3 describes the design and implementation of PubSubCoord; and Section 4 presents conclusions and future work.

2. BACKGROUND

This section provides background information on the underlying individual technologies we use in our solution.

2.1 ZooKeeper

ZooKeeper is a service for coordinating processes of distributed applications [3]. The ZooKeeper service consists of an ensemble of servers that use replication to accomplish high availability with strong consistency and high performance. ZooKeeper provides a watch mechanism to provide a notification service for clients when a specific data node (ZooKeeper data object called *znode* containing its path and data content) or children of a data node is created, updated, or deleted. There are some useful coordination recipes using ZooKeeper often used in distributed applications such as leader election, group membership, and sharing configuration metadata.

Our solution, PubSubCoord, uses ZooKeeper for brokers to discover each other and coordinate routing paths based on events of endpoint discovery detected by edge brokers. Moreover, it is utilized for group membership and leader election in a cluster of routing brokers. The data model of ZooKeeper is like a file system with a simple client API (only read and write). The hierarchical namespace can be used for group membership and in our solution it is used to manage endpoints grouped by topics. Figure 1 shows znode data tree structure of PubSubCoord stored in ZooKeeper servers. The root znode contains three znodes: *topics*, *leader*, *broker*. The *topics* znode contains znodes for created topics and involved endpoints. The *leader* znode is used to elect a leader among routing brokers, and *broker* znode includes znodes for routing brokers where locator information is stored for automatic broker discovery.

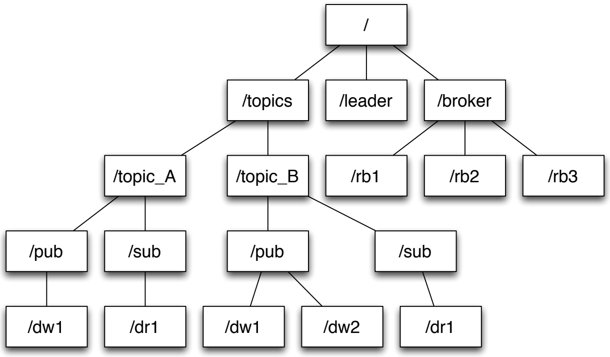

Figure 1: PubSubCoord znode Data Tree Structure

Brokers connect to the servers as clients and create/update/delete znode and also receive notifications by setting watches on interesting znode paths. ZooKeeper provides different modes for znode: ephemeral and persistent. znode with ephemeral mode is automatically deleted when a session of a client which creates the znode to ZooKeeper servers is lost. We utilize this ephemeral mode to easily manage events happened when brokers join or leave our system. As ZooKeeper provides a quorum mode of servers to achieve high availability, the coordination service provided by ZooKeeper avoids single point of failure.

2.2 DDS Routing Service

DDS Routing Service is a service for integrating geographically dispersed DDS systems [6]. It integrates DDS applications across domains (DDS virtual networks), LANs, and WANs. Traditionally, DDS applications only communicate with applications in the same domain in a LAN, but DDS Routing Service enables DDS applications to send and receive data across domains in LANs as well as WANs without any changes on applications. As DDS Routing Service exploits DDS entities for its implementation, it also supports some benefits provided by DDS entities such as a rich set of QoS policies and content-based filtering.

Our solution utilizes DDS Routing Service for brokers to establish DDS data dissemination paths based on routing decisions by coordination logics. As our system is deployed in internet-scale WAN environments, we use TCP communication between brokers for reliable data delivery in geographically dispersed networks. DDS Routing Service supports IP and port translation as well as Transport Layer Security (TLS) for security reasons in WANs. DDS Routing Service can be administered remotely by sending commands (add peers or add topic route) by DDS entities with a special topic, so it is easy to set up routing paths in a programmable way in our solution. Each broker in PubSubCoord sends commands to Routing Service based on its coordination algorithms and event notifications from ZooKeeper.

3. DESIGN OF PUBSUBCOORD

This section describes the design and implementation of PubSubCoord.

3.1 PubSubCoord Architecture

Traditional pub/sub systems form a network with brokers, to which endpoints can be directly connected, to realize a scalable solution. However, it is challenging to maintain routing states for brokers to deliver messages to matching subscribers efficiently. Brokers can be used as intermediate brokers for others, and if a broker fails, it halts not only a service for endpoints connected to this broker but also a service for endpoints connected to other brokers.

To overcome the limitations, PubSubCoord is structured by harnessing a two-tier architecture like the BlueDove system [4]. An entity called edge broker is directly connected to endpoints in a LAN to behave as a bridge to other endpoints located in different LANs. Another entity called routing broker links to edge brokers to deliver data between edge brokers according to assigned topics between geographically dispersed endpoints. Applying this architecture reduces the need for maintaining states for edge brokers to route messages for other brokers in the traditional system and a failed broker does not affect other brokers in a system. Nevertheless, all data traffic goes through routing brokers, and if routing brokers are overloaded or failed, it will impact overall performance of a system.

For that reason, we locate routing brokers in cloud data centers and elastically scale resources of routing brokers depending on their loads. Also, routing brokers form a cluster among themselves and elect a leader for fault-tolerance and load balancing. When an endpoint is created with a new topic, a leader selects a routing broker to assign the topic considering loads of routing brokers in a cluster. When a routing broker fails, an elected leader reassigns topics adopted by the failed brokers to another routing broker to keep providing a service. If a leader fails, routing brokers elect a leader again.

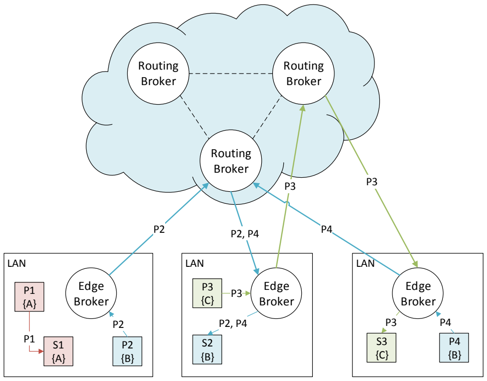

Figure 2: PubSubCoord Architecture

Figure 2 shows an example applying the PubSubCoord architecture. P$x\{y\}$ is a publisher identified with x interested in topic y. As there are no other endpoints interested in topic A other than *P1* and *S1*, they only communicate in a LAN with UDP multicast as a default transport for scalability and low latency. *P2*, *P4*, and *S2* are interested in topic B and these are located in different networks. They communicate with each other over a routing broker responsible for topic B. A network transport between brokers is configurable, and TCP unicast is used as default transport because transmissions occur in WANs or a cloud data center which may cause high rate of packet loss.

3.2 Implementation of PubSubCoord

Algorithms 1 and 2 describe the pseudo code for event callback functions in edge brokers and routing brokers, respectively. Callback functions are invoked by either DDS built-in DataReader for endpoint discovery or watch notifications by ZooKeeper.

Algorithm 1 Edge Broker Callback Functions

```
function ENDPOINT CREATED($ep$)
    create_znode ($ep$, $ep_{path}$)
    if ! $topicMultiSet$.contains($ep_{topic}$) then
        $node\_cache$ = create_node_cache ($ep_{topic}$)
        set_listener ($node\_cache$)
        $routingService$.createTopicRoute($ep_{topic}$)
    $topicMultiSet$.add($ep_{topic}$)
function ENDPOINT DELETED($ep$)
    delete_znode ($ep_{path}$)
    $topicMultiSet$.delete($ep_{topic}$)
    if ! $topicMultiSet$.contains($ep_{topic}$) then
        delete $node\_cache$
        $routingService$.deleteTopicRoute($ep_{topic}$)
function TOPIC CACHE LISTENER($nodeCache$)
    $rb\_locator$ = $nodeCache$.getData()
    if ! $rbPeerList$.contains($rb\_locator$) then
        $rbPeerList$.add($rb\_locator$)
        $routingService$.addPeer($rb\_locator$)
```

Each callback function for edge brokers is invoked when the following events occur:

- *Endpoint Created* - It is invoked when an endpoint (publisher or subscriber) in a network is created and activated by a built-in DDS Data Reader for endpoint discovery.
- *Endpoint Deleted* - It is invoked when an endpoint (publisher or subscriber) in a network is deleted. This callback function is activated by a built-in DDS Data Reader for endpoint discovery.
- *Topic Cache Listener* - It is invoked when a topic znode managed by an edge broker is created, deleted, or updated and activated by ZooKeeper client API.

Endpoint Created callback function first creates a znode for a created endpoint to notify this discovery event to routing brokers. If a topic interested by the created endpoint has not appeared in an edge broker before, a znode cache and its listener for the topic is created to receive notifications for the znode update. When the znode is updated, it triggers the*Topic Cache Listener* callback in Algorithm 1.

We used the Curator framework [1], a high-level API that simplifies using ZooKeeper, and it provides useful recipes such as leader election and caches. We used the cache recipe to reserve data accessed multiple times for fast data access and reducing loads on ZooKeeper servers.

Endpoint Deleted callback function deletes the znode for the existing endpoint to notify a routing broker, and deletes it from a multi set for topic. Then, it checks the multi set contains the topic. If the topic is contained in the multi set, it means other endpoints are still interested in the topic. If it is empty, the cache and its listener need to be removed as there are no endpoints interested in the topic.

In *Topic Cache Listener* callback function, each topic znode stores a locator of a routing broker which is responsible for the topic. The locator of a routing broker is added to a Routing Service running in an edge broker to establish a communication path between an edge broker and a routing broker.

Algorithm 2 Routing Broker Callback Functions

```
function RB CACHE LISTENER($nodeCache$)
    $topic\_set$ = $nodeCache$.getData()
    for $topic$ : $topic\_set$ do
        if ! $cacheList$.contains($topic$) then
            $children\_cache$ = create_children_cache ($topic$)
            set_listener ($children\_cache$)
            $cacheList$.add($topic$)
function ENDPOINT CACHE LISTENER($childrenCache$)
    $cache\_data$ = $childrenCache$.getData()
    $eb\_locator$ = $cache\_data$.getLocator()
    $topic$ = $cache\_data$..getTopic()
    switch $cache\_data$.getEventType() do
        case $Child\_Added$
            if ! $ebPeerList$.contains($eb\_locator$) then
                $ebPeerList$.add($eb\_locator$)
                $routingService$.addPeer($eb\_locator$)
            if ! $topicList$.contains($topic$) then
                $routingService$.createTopicRoute($topic$)
            $topicMultiSet$.add($topic$)
        case $Child\_Deleted$
            $topicMultiSet$.delete($topic$)
            if ! $topicMultiSet$.contains($topic$) then
                $ebPeerList$.delete($eb\_locator$)
                $routingService$.deleteTopicRoute($topic$)
```

Each callback function for routing brokers is invoked when the following events occur:

- *RB Cache Listener* - It is invoked when a znode for a routing broker is updated and activated by the ZooKeeper client API.
- *Endpoint Cache Listener* - It is invoked when children of a znode for an assigned topic is created, deleted, or updated and activated by ZooKeeper client API.

In *RB Cache Listener* callback function, a znode for a routing broker stores a set of topics assigned by a cluster leader. When the topic set is updated, it applies changes by creating a cache and its listener for endpoints interested in assigned topics.

When an endpoint is created or deleted, edge brokers create or delete znodes for endpoints and these events will notify *Endpoint Cache Listener* callback function in routing brokers by ZooKeeper's watch mechanism. The znode cache in *Endpoint Cache Listener* callback function stores a locator of an edge broker and its topic name. The event can be creation or deletion, so it needs to be differentiated by looking at an event type. If it is the creation events, a locator of an edge broker needs to be added to DDS Routing Service running in this routing broker if it does not exist. After that, it requests the Routing Service to create a route for the topic from a routing broker to an edge broker. If an event type is deletion, it has to delete a locator and a topic route.

4. CONCLUDING REMARKS

OMG DDS has been used successfully for many mission critical systems when these are deployed in the same network. However, as a system scales up, it requires integrating pub/sub endpoints in different networks to realize internet-scale, QoS-enabled pub/sub systems. Realizing internet-scale DDS is hard for a variety of reasons. This paper presents preliminary work on a cloud-enabled coordination service for internet-scale DDS applications that can support scalability, fault-tolerance, and endpoint mobility.

Our future work primarily includes complexity analysis and empirical evaluation to validate the scalability and low-latency of our solution. Specifically, we will measure (1) end-to-end latency and throughput of endpoints located in different networks, (2) discovery time of moving endpoints from an edge broker to another, and (3) response time of coordination events from ZooKeeper with increasing number of endpoints and brokers. We also plan to investigate automatic configurations of content-based filtering on brokers to reduce network and computation overhead. Additional dimensions of future work involves research on adaptive overlay network according to the deadline information provided by endpoints since DDS endpoints can allow users to define deadline values and these values can be used as hints by brokers to structure a deadline-aware overlay network.

5. ACKNOWLEDGMENTS

This work is supported in part by NSF CAREER CNS 0845789. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF.

6. REFERENCES

[1] Apache. Apache curator. `http://curator.apache.org`, 2014.

[2] A. Hakiria, P. Berthoua, A. Gokhalec, D. C. Schmidtc, and G. Thierrya. Supporting end-to-end scalability and real-time event dissemination in the omg data distribution service over wide area networks. *Submitted to Elsevier Journal of Systems Software (JSS)*, 2013.

[3] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, volume 8, pages 11–11, 2010.

[4] M. Li, F. Ye, M. Kim, H. Chen, and H. Lei. A scalable and elastic publish/subscribe service. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1254–1265. IEEE, 2011.

[5] OMG. The data distribution service specification, v1.2. `http://www.omg.org/spec/DDS/1.2`, 2007.

[6] RTI. Rti routing service user's manual. `http://community.rti.com/rti-doc/510/RTI_Routing_Service_5.1.0/doc/pdf/RTI_Routing_Service_UsersManual.pdf`, 2013.