Resilient and Preserving Dissemination of Events in a Large-Scale Event Notification Service System

Chit Htay Lwin\(^1\)  Hrushikesha Mohanty\(^1\)  R. K. Ghosh\(^2\)  Goutam Chakraborty\(^3\)

Department of CIS\(^1\)  Department of CSE\(^2\)  Department of SCS\(^3\)

University of Hyderabad, India  IIT Kanpur, India  Iwate Perfectural University, Japan

\{mhcsrcs, hmcs\}@uohyd.ernet.in  rkg@cse.iitk.ernet.in  gotum@soft.iwate-pu.ac.jp

Abstract

Event Notification service system is a data dissemination technology which asynchronously notifies consumers whose interests match with the events published by producers. Fault-tolerance is important for a large-scale event notification service system as link or node failures usually occur in a wide-area network. In this paper, we first describe architecture of event notification service system for large scale network to minimize the size of routing entities and to reduce latency of notification delivery to consumers. We then present a replication algorithm based on primary-back replication so that event notification service system is resilient to failures of event servers and links between them and maintains dissemination of events. The replication technique used in proposed architecture can minimize the portion of the system affected by these failures.

1. Introduction

As an event notification service (ENS) system has multicast characteristic, it allows a producer to send an event to many interested consumers with only one publishing operation. Event servers implement the entire functionality of an event notification service system and these are interconnected as a distributed network and scalable. An ENS system can cope with dynamically changing operational environment where producers and consumers frequently disconnect and reconnect because it has loose coupling among producers and consumers. Hence, an event notification service system is a good candidate for a variety of e-commerce and m-commerce applications. In [6], it shows that event-based cooperation models have better supports for a variety of large-scale e-commerce applications.

In large-scale network, subscriptions of routing table increases at each event server when number of consumers become large. When an event comes in, each event server matches the event with all subscriptions of its routing table to forward to neighboring event servers. Hence, the event is matched with subscriptions at all event servers which are located along the path between the producer who publishes the event and an interested consumer. It increases matching time and causes some delay in delivering events to interested consumers in large-scale network. Hence we present architecture which is scalable and efficient to deal with failures of event notification service. The architecture can reduce size of routing tables by forwarding subscriptions only within a region and it can reduce matching delay at event servers by sending published notifications to reach all regions.

Fault-tolerance is an important feature in a large-scale event notification service system as link or node failures are frequent in wide-area networks. An event notification service system can have the following failures:

- Link failure: A link failure is an inability of an event server to contact one of its neighbors. This can happen by a failure at the IP routing level.
- Event server failure: If an event server fails, its all neighbors cannot contact it. An event server may also be disconnected from one of its neighbors because the physical link connecting the two has failed. One possible solution is event server that detected the failure establishes new paths without including the failed event server.
- Consumer/Producer failure: If a consumer/producer fails, it has no contact to its local event server. After a consumer fails, the consumer’s event server will unsubscribe the subscriptions of the consumer after a stipulated time. A producer cannot publish notifications during its failure and may resume publishing these notifications on its recovery.

Fault-tolerance mechanism has to be part of ENS system so that isolated network or component failures do not affect the entire system [11]. In [13], it states that a fault-tolerant system should continue functioning perhaps in a degraded form, when faced failures. The degradation can be in performance, in functionality, or in both. Hence fault tolerance and scalability are related each other. In event notification service systems, a failed event server/link can cause a break in event dissemination tree.
Hence, the event server that detects a failure needs to re-route subscriptions and notifications that previously went via the failed event server. For resilience of event server failure, the technique of replication of subscriptions may be applied.

In ENS system the routing of an event is dependent on event content rather than particular destination address and each event server is only aware of its neighboring event servers through which events propagate. Hence, existing replication technologies [7, 8] cannot be used directly. Hence, we describe a replication algorithm making the event notification service system resilient to failures in proposed architecture for large-scale network.

The rest of the paper is structured as follows. Section 2 describes related work concerning failures in ENS systems. Section 3 presents an architecture for a large scale event notification service system. Section 4 describes a basic concept of replication technique. Replication algorithm for proposed architecture is described in section 5 and section 6 concludes the paper.

2. Related work

Herald event notification service [2] uses many replicated servers in different locations to execute some or all work of a rendezvous point for scalability and fault-tolerance. It provides a degree of fault tolerance that allows clients (both publishers and subscribers) to interact with any of the replicas of a rendezvous point for any operations. Hermes [11] uses overlay routing network which is a logical application-level network built onto the physical network topology. Link and node failures are dealt with transparently by the overlay network. It also replicates rendezvous nodes so that a consumer is still able to receive events coming from producers attached to a different replica. Hence [2, 11] can provide fault tolerance caused by rendezvous nodes by connecting one of replicas. But in case of link failure, producers/consumers continue publishing/receiving events from only partitioned network they are located.

[4, 9] describes reconfigurations that involve the removal of a link and the insertion of a new one, thus keeping the dispatching tree connected. The approach described in [4, 9] is based on subscription (to remove a link) and subscription (to insert a new link). It takes into account reducing reconfiguration messages and it enables that insertion of subscriptions and removal of subscriptions between endpoints of removed link and new link. The subscriptions triggered by the appearance of a link are issued immediately, while the unsubscriptions due to a link removal are issued only after a predefined delay. An approach which is resilient to join and leave of event servers is presented in [1]. After an event server leaves, new connections are established and the state of the routing tables is kept consistent removing the subscriptions that were hosted by the leaving event server and adding the newly created paths. Each event server maintains a list of neighbors for any of its neighbors. The list is updated whenever a new event server joins or leaves. Hence if an event server fails, one of the neighbors of failed event server requests a new connection to an event server in its list.

3. Architecture and routing strategy

Most of the ENS system [4, 10, 15] use overlay networks for interconnection of event servers. In our architecture, event servers are connected as overlay network.

The network of event notification service system is divided into a few big regions which have separate networks composed of event servers. Each region has an event server that is responsible for communicating other event servers located in other regions. We call such an event server as boundary event server (BES). In each region, a center event server (i.e., center node of a graph) is selected as BES to minimize the latency of delivering notifications published from other regions to event servers in its region. BESs can be linked together in any interconnection topology such hierarchical or peer-to-peer. Other event servers of a region can also be interconnected in any topology within a region depending on the characteristics of the applications and it does not depend on interconnection topologies of other regions. Siena [3] introduced hybrid architecture to use different architectures at different levels of network (for example, local area and wide area) for different levels of administration.

The three layers of networks in the architecture are illustrated in Figure 1. The bottom layer is the physical layer network with servers and links. The middle layer constitutes the peer-to-peer overlay network interconnected event servers, each one serving some subset of the clients of the service. The top layer is network of boundary event servers where each region has a boundary event server.

3.1. Subscription and notification

Each event server (including BES) maintains a routing table for propagation of subscriptions and notifications. And a routing table (RT) is a collection of tuples \{<subscription>, <consumers>, <incoming servers>, <outgoing servers>\);
In routing tables, the consumers associated to a subscription \( S \) mean the local consumers who subscribe subscription \( S \). Incoming servers of subscription \( S \) are the event servers from which subscription \( S \) is received, and outgoing servers of subscription \( S \) are the event servers to which \( S \) is forwarded to. If a notification \( N \) is received from a producer, local event server matches \( N \) with all subscriptions stored in RT. The results are:

1. a set of local consumers serviced by itself and
2. a set of event servers to which \( N \) needs to be forwarded.

A notification \( N \) matches a subscription \( S \) if \( S \) covers \( N \) (\( N \subseteq S \)). If RT has subscriptions \( S \) matched with \( N \) (i.e. \( \exists S \in \text{RT} \mid N \subseteq S \)), the local event server delivers \( N \) to the consumers \( \{ \text{RT.S.C.ID} \} \) and forwards \( N \) to event servers \( \{ \text{RT.S.IS} \} \).

### 3.2. Routing strategy

Routing algorithm for event dissemination determines the overall scalability of ENS system. We use a hybrid routing strategy that is more scalable than using one of existing routing strategies because our system does not require global broadcast of subscriptions or notifications. In boundary event server network, each boundary event server only forwards notifications. Notification forwarding takes place to boundary event servers instead of all event servers. Hence we use notification forwarding at global network level and subscription forwarding at region network level as follows.

#### 3.2.1. Notification forwarding in BES network

When an event server receives a notification from its local producer, it sends the notification to its BES. On receiving this notification, each BES forwards it to reach all other BESs. A BES then matches the notification against subscriptions of its routing table to forward the notification to neighboring event servers and consumers in its region.

#### 3.2.2. Subscription forwarding in region network

Subscription forwarding strategy is used in each region. When an event server receives a subscription from a consumer or its neighboring event servers, it forwards the subscription to neighboring event servers. Subscriptions of consumers of a region are not forwarded to other regions as boundary event server of this region can receive all notifications published from other regions. Each boundary event server then matches the notifications with subscriptions of its routing table and forwards them to its neighboring event servers only if at least an interested consumer exists in its region.

### 3.3. Advantages

The proposed architecture has the following advantages.

1. It improves reliability. If a link or an event server fails in a region, it does not affect other regions. Notifications can reach to all regions if BESs and links between them do not fail. Additional event servers can be added in order to improve performance or reliability of the system in a region without affecting performance and functionality of other regions.

2. It reduces the size of routing tables. In large-scale network, subscriptions of a routing table increases at each event server when number of consumers become large. Proposed architecture can reduce the size of routing table because subscriptions are only forwarded to event servers within a region. Hence, each event servers only maintains a routing table with subscriptions of consumers in the same region.

3. It reduces matching time (overhead) at event servers. In existing event notification service systems [3, 5, 14, 15], when an event comes in, each event server matches the event with all subscriptions of its routing table to forward the event. Hence, the event is matched with subscriptions at all event servers which are located along the path between the producer who publishes the event and an interested consumer. In our architecture, BES forwards notifications to reach all BESs. Hence it reduces the time and cost of matching at event servers as notifications published from other regions reach to a region without matching process.

### 4. Replication technique for fault-tolerance

The data replicated at backup event server is a routing table (RT) as described in section 3.1. A routing table of an event server and virtual routing table replicated at backup event server are necessary to be consistent to avoid loss or duplication of notifications.

**Figure 2. Managing failure of event server ES\(A\)**

In replication approach if an event server ES\(A\) does not crash, neighboring event servers of ES\(A\) connect to ES\(A\). If neighboring event servers of ES\(A\) suspects ES\(A\) has failed, they connect to ES\(n\) which is backup event server of ES\(A\). ES\(n\) maintains a virtual routing table of ES\(A\) (VRT\_A) besides its own routing table RT\_B. Hence RT\_A and VRT\_A are necessary to be consistent.
If a subscription is lost in a routing table, the consumer who subscribed the subscription will not receive notifications which match the subscription. Hence, an event server updates on demand (i.e., whenever a new subscription is inserted in its routing table) to an event server. An event server sends acknowledgement for a subscription after inserting the subscription in the routing table. Hence if an event server fails before a subscription is inserted in the routing table, the event server which sent the subscription will resend it to backup event server.

Our approach is based on primary-backup replication model as described in [7, 8, 12]. We assume that failure of a boundary event server can be eventually detected by its neighbors. For example, a heartbeat protocol ensures that the neighboring event servers are reachable and alive [11]. To detect a link or an event server failure, a handshaking protocol [13] can also be used. Suppose ESₐ is a backup event server of ESₐ as shown in Figure 3. ESₐ maintains its routing table RT_B and virtual routing table VRT_A which is shown in Figure 3 as an example.

**ESₐ has not failed (normal condition):** ESₐ enables its RT_A and neighbors of ESₐ are connected to it. ESₐ disable VRT_A which is backup routing table of RT_A and it only executes its own processes with RT_B. Whenever a subscription is inserted in RT_A, ESₐ sends the subscription to ESₐ to update VRT_A.

**EST_A = Virtual routing table of ESₐ replicated at ESₐ**

<table>
<thead>
<tr>
<th>S</th>
<th>IS</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>ES₁</td>
<td>ESₐ</td>
</tr>
<tr>
<td>S₂</td>
<td>ESₐ</td>
<td>ESₐ</td>
</tr>
</tbody>
</table>

**RT_B = Routing table of event server ESₐ**

<table>
<thead>
<tr>
<th>S</th>
<th>C_ID</th>
<th>IS</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>C₁</td>
<td>ESₐ</td>
<td>ESₐ</td>
</tr>
<tr>
<td>S₂</td>
<td></td>
<td></td>
<td>ESₐ</td>
</tr>
</tbody>
</table>

Figure 3. Representing routing tables at ESₐ

**ESₐ has failed:** ESₐ enables VRT_A to execute the functions of ESₐ. Neighboring event servers of ESₐ connect to ESₐ instead of ESₐ. When ESₐ receives a subscription/notification from neighboring event servers of Sₐ, ESₐ first executes (i.e., covering test or matching) the subscription/notification with RT_B. If the subscription/notification needs to be forwarded to ESₐ, ESₐ then execute it with VRT_A. For example if a notification N matches Sₐ of RT_B and N needs to be forwarded to ESₐ, ESₐ then matched with subscriptions of VRT_A. Similarly, if subscriptions or notifications come from neighboring event servers of ESₐ, ESₐ first executes them with VRT_A then with RT_B. When neighboring event servers of ESₐ detects ESₐ has recovered, they try to connect to ESₐ and inform ESₐ to update RT_A with VRT_A.

5. Algorithm

In this section we present our replication algorithm used in the architecture described in section 3. A boundary event server (BES) in a region is a primary and it is the only event server to interact with other BESs. Each BES has a backup event server which is one of its neighboring event servers in the same region. The difference between replication algorithm in an event notification service system and existing primary-backup approach is that backup event server in event notification service system executes its own processes although the BES does not fail. Hence, a backup event server needs to maintain both of its routing table and virtual routing table of BESs.

Suppose ESₐ is a boundary event server and ESₐ is a backup event server of ESₐ. As shown in Figure 5, ESₐ is interconnected with other BESs of other regions. The data structures used in the algorithm are as follows:

RT_A = Routing table of event server ESₐ

VRT_A = Virtual routing table of ESₐ

Sₐ = a set of event servers which has failed to connect

RegionNe (ESₐ) = Neighboring BESs of ESₐ

LocalPro (ESₐ) = all local producers of ESₐ

LocalCon (ESₐ) = all local consumers of ESₐ

Neighbor (ESₐ) = Neighboring event servers of ESₐ

LocalES (ESₐ) = Event servers in the region of BES ESₐ

5.1. Processes of a boundary event server

A boundary event server ESₐ needs to execute the processes as described in Figure 6. For simplicity, we assume the boundary event server does not have local clients. ESₐ can receive a subscription S from neighboring event servers (Neighbor (ESₐ)) (Algorithm 1. S₁, S₃).

```
1. //The following is executed if ESₐ is not suspected of failure
   result ← alive
   //ESₐ is alive
   repeat
   S₁: On receiving Subscribe (S) from ESₐ ∈ Neighbor (ESₐ),
      Forward (S);
      //...forwards S after checking RT_A
      Send Update (S) to ESₐ;
      //update S in VRT_A at ESₐ
   S₂: On receiving Update (S) from ESₐ
      Insert S into RT_A;
```

Figure 4. Replication of BESs in multiple regions
N1: On receiving \textit{Forward} (N) from ES\(_i\) ∈ Neighbor (ES\(_a\))
   Match (N);
N2: On receiving \textit{Broadcast} (N) from ES\(_i\) ∈ RegionNe (ES\(_a\))
   \textbf{For} each ES\(_k\) where ES\(_k\) ∈ (RegionNe (ES\(_a\)) – ES\(_i\))
   \hspace{1em} \textbf{if} (ES\(_k\) \notin S) \hspace{1em} //if ES\(_k\) suspect ES\(_i\) has failed…
   \hspace{1em} Send \textit{Broadcast} (N) to ES\(_k\)
   \textbf{else}
   \hspace{1em} Send \textit{Broadcast} (N) to Backup\(_i\)
\textbf{endFor}

C1: On receiving \textit{UpdateOver} () from ES\(_b\)
   Enable RT\(_A\);
C2: On receiving \textit{VirtualRTO\(_i\)} () from ES\(_b\) // Backup\(_i\)
   Disable RT\(_A\);
C3: if (ES\(_i\) \notin S) where ES\(_i\) ∈ RegionNe (ES\(_a\))
   \hspace{1em} //if ES\(_i\) knows ES\(_a\) has recovered…
   \hspace{1em} send \textit{Alive} (ES\(_i\)) to ES\(_a\) //Backups of RegionNe (ES\(_a\))
C4: if (ES\(_i\) \notin S) where ES\(_i\) ∈ RegionNe (ES\(_a\))
   \hspace{1em} //if ES\(_i\) knows ES\(_a\) has failed…
   \hspace{1em} send \textit{Suspect} (ES\(_i\)) to ES\(_a\) //Backups of RegionNe (ES\(_a\))
\textbf{until} (result = suspect)

\textbf{II.} // The following is executed if ES\(_a\) is in suspect condition
\textbf{repeat}
S3: On receiving \textit{Subscribe} (S) from ES\(_i\) ∈ Neighbor (ES\(_a\))
   Send \textit{ConnectBackup} (ES\(_b\)) to ES\(_a\) //to connect to ES\(_b\)
C5: On receiving \textit{Activate} (ES\(_a\)) from ES\(_b\)
   Send \textit{Fetch} () to ES\(_a\); //to retrieve data from ES\(_b\)
   \hspace{1em} result ← alive
   \textbf{until} (result = alive)
\textbf{endFor}

\textbf{Figure 5. Algorithm 1 executed at ES\(_a\)}

\textbf{Figure 6. Algorithm 2 executed at ES\(_a\)}

ES\(_a\) can receive a notification \(N\) from:
(1) neighboring event servers (Neighbor (ES\(_a\)))
(2) neighboring BESs (RegionNe (ES\(_a\))) and
(3) event servers in this region (LocalES (ES\(_a\)));
These event servers send notifications received from their local producers to BES.

\textbf{5.2. Processes of an backup event server}

\(\text{If ES}_b\) does not fail, \(\text{ES}_b\) will execute itself without the processes of \(\text{ES}_a\) (Algorithm 2. S1, S2, and N1, N2) like other event servers. If \(\text{ES}_a\) fails, \(\text{ES}_b\) needs to execute its own processes and processes of \(\text{ES}_a\). The algorithm executed at \(\text{ES}_b\) is described in Figure 6.

\(\text{ES}_b\) can receive a subscription \(S\) from:
(1) neighboring event servers (Neighbor (ES\(_a\)))
(2) local consumers (LocalCon (ES\(_b\)))
(3) neighboring event servers (Neighbor (ES\(_a\)))
\(\text{ES}_b\) can receive a notification \(N\) from:
(1) neighboring event servers (Neighbor (ES\(_a\)))
(2) local producers (LocalPro (ES\(_a\)))
(3) neighboring event servers (Neighbor (ES\(_a\)))
(4) neighboring BESs (RegionNe (ES\(_a\)))
(5) event servers in this region (LocalES (ES_i)).
LocalES (ES_i) = LocalES (ES_0)

6. Conclusions

In this paper, we presented an architecture for large scale network of event notification service system and a replication algorithm for this architecture. The replication algorithm makes the system resilient to failures of event servers and links between them. The replication algorithm relies on the assumption that failure of a boundary event server can be eventually detected by its neighboring event servers and then they can connect to the backup event server.

In our architecture subscriptions are only forwarded to event servers in a region and thus subscription forwarding and routing tables of event servers are not related to subscriptions of other regions. Notifications (events) are forwarded to reach every boundary event server without matching to minimize the propagation time between producers and consumers. Hence link or event server failures in a region do not affect to other regions. The replication algorithm maintains data consistency of routing tables of boundary event server and backup event server, and can avoid loss and duplication of events when a boundary event server fails. The algorithm is only executed for boundary event servers and thus it is not costly in the system as the number of boundary event servers is very less than the total number of event servers of the whole network. The approach we presented in this paper requires further investigation for evaluation of costs and benefits.

Reference