SESDAD - A Prototype Publish-Subscribe System with Ordering Guarantees

Luís Gonçalves 58321
João Soares 73959
Bernardo Rodrigues 73916

Abstract

Publish-Subscribe systems offer a group communication service in which a message is addressed to a group and subsequently delivered to all members of that group. The SESDAD Publish-Subscribe system proposed in this document aims to offer such a service in a decentralized fashion with guarantees regarding message ordering. The system was tested with various configurations and analyzed to determine it's performance in varying conditions.

1. Introduction

The SESDAD Publish-Subscribe system aims to deliver a group communication service, with a topic-based subscription model, while also guaranteeing three possible message ordering paradigms. These are no-order, in which there is no guarantee about message ordering, FIFO (First-In-First-Out), in which each message is delivered by the order in which the original publisher published the messages, and total order, which guarantees that each subscriber receives the messages in the same order.

A fixed tree-based topology for the overlay network was chosen, this simplification allows for a simpler reasoning about the routing algorithms that were implemented in the system. Each node in the tree, refered from now on as a "site" contains a broker, responsible for propagating the message to other brokers and delivering messages to the relevant subscribers, and also possibly contains multiple publishers or subscribers.

The system supports two different routing configurations, the first uses a flooding mechanism in which all messages are propagated through the whole tree-based overlay network, the second one filters messages according to which paths lead to interested subscribers.

In order to easily test the system a "Puppet Master" application was developped to essentially control the system and provide a centralized message logging.

The next section summarizes the design and discusses some dificulties of implementation.

2. Design

The SESDAD system belongs to the structured p2p class of distributed systems, using a tree-based topology overlay network. Each node in the tree is a site containing one broker process and which might contain subscriber or publisher processes. These publishers or subscribers are directly connected to the site's broker and only send messages to it. Each broker connects to the parent site's broker and each children broker. Each published message that is to be routed goes through the broker Event Routing processing, which propagates the message according to the configuration options specified. This processing might also send a log message to the "Puppet Master" depending on the logging level defined in the configuration file.

The message propagation is "asynchronous", meaning the initial broker does not wait for the message to be fully propagated through the system before returning, instead it generates a new thread for each processing of the message that is needed. So, a thread is started to propagate the message, another one to send the log messages and another one to process the delivery of the message to interested subscribers, so that the calling broker which is running the thread that propagates the message can continue without waiting. This causes some issues in the ordering of messages that need to be addressed.

The next sections describe those issues and how we addressed them by providing a summary of the various configuration options' implementation.

2.1. Routing Configurations

The routing mechanism is divided in two configurations:

Flooding: The flooding algorithm adopted is very simple and takes advantage of the tree topology simplification. When a broker receives a message to route it sends that message to every neighbor except the last hop. Since the topology is tree-based there is no possibility for loops in the topology graph, avoiding the propagation of repeated messages through the system.

Filter: The filter algorithm mantains a set of "neighbor filters", essentially a routing table, containing all topics, and

number of subscribers for those topics, that should be routed to each neighbor, along with all publisers, and their topics, that can be reached through each link. The mechanism is based on the advertisement approach, striving to guarantee a coherent state between new subscriptions, unsubscriptions and new topics being published.

There are three kinds of advertisement messages: Subscription Advertisements, Publication Advertisements and Unsubscription Advertisements, along with some support messages, Filter Update and Sequence Update, the latter is only used when the ordering configuration is FIFO.

Each advertisement message locks the message processing in each broker until the advertisement is resolved by the system. All advertisement messages are propagated to the root of the tree. This is important in order for the broker in which the Publication and Subscription Advertisements intersect to correctly propagate the necessary Filter and Sequence Update messages.

Subscription advertisement: Upon receipt of a subscription advertisement the broker increments the subscription count of the topic on the routing table indicating an interest for the topic through the link. If the broker already had another link with interest in that topic or if a subscriber for that topic is already connected then the messages for this topic are already being sent to the broke. If not, it then checks if he has knowledge of a publisher for that topic. If a publisher is known through a link besides the parent broker, it then sends a Filter Update message in the direction of the publisher.

Publication advertisment: Upon receipt of a publication advertisement the broker checks if he knows of any subscription for that topic through a link other than to parent, if yes then it sends a Filter Update message to the last hop.

Unsubscription advertisement: An unsubscription advertisement decrements the count of subscribers for a topic through the link from which the message was received. It then propagates the message through the links with known publishers for that topic.

Filter Update: A filter update message is only received by child brokers who know a publisher for the topic through a link, indicating that an interest in the given topic is requested from the parent broker. The message receiver increments the interest count of the topic on the link from which the message was received and checks if any publishers for that topic are known to his children, if yes, then a Filter Update message is sent to them.

2.2. Ordering Configurations

The ordering mechanism is divided in three configurations:

NO ORDER: When a broker receives a message it immediately delivers it to the directly connected interested subscribers for the message topic. There is no difference in the routing mechanisms for this type of message ordering.

FIFO: Each publisher assigns a sequence number to the published messages according to the order by which they are processed by it. Each broker has a wait queue containing messages received. Upon receiving a publication message the broker first adds it to the wait queue, then it iterates over the wait queue to find the message with a sequence number corresponding to the publisher delivered messages sequence number. If such a message is found it is sent to all interested subscribers and removed from the wait queue. This type of ordering warrants differences in the filter routing mechanism due to the sequence numbers, a descriprion of the changes to the filter mechanism, and a descciption of the Sequence Update message is summarized bellow.

Filter Update: When a broker now receives a filter update message it has to verify the sequence number for all known publishers of that topic. If the broker does not know the sequence number, because thre was previously no interest in any topic from the publishers that published the given topic, it propagates the message as described earlier. If it does know the sequence number, it sends a sequence update message containing the current sequence number for each publisher that publishes the given topic.

Sequence Update: Upon receiving a sequence update a broker needs to propagate the sequence number update if the current tracking of the sequence numbers is updated, meaning the current sequence number for that publisher is lower than the one received in the sequence update message. The propagation is implemented by sending a sequence update message to all neighbors interested in the topic that generated the sequence update message, which ultimately is the topic for the filter update message that originated the sequence update. The sequence update message should propagate from the first broker who receives a filter update message, who knows the sequence number, up the tree until the broker which knows no other links other than the sequence messages last hop which have interest in the topic.

TOTAL: The root broker is selected as the sequencer for all messages pertaining publications. This sequencer is responsible for attributing sequence numbers to the publishers publications. When a broker receives a publication from a locally connected publisher it first propagates a sequence request message up the tree to the root broker, serving as the sequencer, which sends the sequence number for that message as a return value. This first exchange of messages for each message published is synchronous, meaning that the broker first receiving the publication waits for the sequence request message to be fully propagated to the sequencer in order to receive a consistent sequence number.

The delivery mechanism is similar to the FIFO ordering except that each broker now keeps track of the last sequence

number delivered.

Using the flooding mechanism there is no problem with this simple approach, although for the filter routing mechanism some changes are needed which were not implemented in the delivered version of the SESDAD distributed application.

3. Implementation Details

In this section a detailed explanation of the implementation of the design decisions is presented. The algorithms are analyzed and a brief reasoning about their function is discussed.

The delivered SESDAD application was written in C# using the .net Remoting library for inter-process communications.

Each publication message originating from the publishers contains the publisher address, the sequence number of the message, the message topic and the message itself represented as a string. In addition to this the address of the last hop is also sent with virtually every message passed through the system.

3.1. Flooding algorithm

The flooding algorithm, represented in algorithm 1, relies heavily on the fact that the overlay network graph is acyclical, hence the messages have to be propagated to every connected broker except the last hop to avoid repeating messages in infinite loop over the overlay network.

Algorithm 1: Flooding algorithm

```
e = received message;
if parent != null and parent.address !=
  lastHopAddress then
    parent.flood(e);
end
foreach child in children set do
    if child.address != lastHopAddress then
        child.flood(e);
    end
end
```

3.2. Filtering algorithm

The filtering algorithm also never sends messages back to the last hop. However a new set of messages is introduced to allow the system to perform an efficient routing of messages and not send messages through links which have no interest in the message's topic.

Before any message is propagated from the original broker(the one belongging to the publisher's site) through the network the broker first verifies if the message is from a newly published topic for that publisher and starts the advertisement phase of the publication. This phase chains a Publication Advertisement message to the root broker. When this phase is complete the message routing mechanism can perform it's duties and route the message.

It is worth noting that, altough there is a Publication Advertisement mechanism, there is no "Unpublication" Advertisement implemented in the delivered SESDAD application, meaning there is room for improvement in the system in this regard.

Depending on the ordering configuration chosen, it might be necessary to send messages through links which do not have a direct interest in the topic but are interested in other topics that are published by this publisher. The message has to be sent in order to preserve the coherence of the sequence numbers for that publisher.

This mechanism is represented in algorithm 2.

Algorithm 2: Filter algorithm

```
e = received message;
foreach filter in neighbor filter set do
    if filter.hasInterestInTopic(e.topic) then
        filter.neighbor.sendMessage(e);
    end
    else
        foreach publisherTopic in Publisher topics do
            if filter.hasInterestInTopic(publisherTopic)
              and publisherTopic != e.topic then
                filter.neighbor.sendMessage(e);
            end
        end
    end
end
```

3.3. Advertisements implementation

The idea of advertisements is to allow the system to update it's filters and sequence numbers, in order to do this there is a need to stop message processing between brokers to avoid desynchronization of sequence numbers.

A broker must receive the sequence number for the publisher he needs to know about before any other messages for that publisher arrive. This poses a problem in the case of Subscription Advertisements.

Upon receiving a subscription advertisement the broker must ensure that no message is processed before the broker that needs the sequence number gets the message. This functionality was implemented through critical section

locks on the advertisements and message processing on the brokers.

Since the advertisement mechanism is supposed to be synchronous, we have to be carefull on the locks chosen. For example, when a Subscription Advertisement is received it locks the message processing queue, however it might need to send a Sequence Update message back to the sender, which must not lock the queue because it is already locked and a deadlock occurs, since the messages are synchronous, and the system stalls. Thus these messages lock on another variable in order to avoid the deadlock.

It is worth noting that since the Subscription Advertisment is synchronous the message processing will only resume once the sequence update has been received by the required brokers.

Another problem arises from the need to send a Sequence Update when an Update Filter message is received. The broker receiving the message knows the sequence number of a publisher publishing the requested topic and sends the Sequence Update message back to the sender. Thus these two messages must not lock each other, and are not synchronous, meaning that only guarantees about message delivery are needed. The brokers only need to know that the other broker has received the message and carry on. If the broker receives multiple Sequence Updates or Filter Updates they are mutually-exclusive through locks and guarantee coherency while updating internal structures.

Algorithms 3, 4 and 5 represent the advertisement mechanism.

Algorithm 3: Subscription Advertisement

```
add topic to last hop filter;
if new topic added to interests in filter for last hop
 then
    foreach publisher in known publishers do
        if publisher publishes topic and there is no
          previous interest in any of the publisher's
          other topics then
            send Sequence Update to last hop;
        end
    end
end
foreach filter in neighbor filter set do
    if publishers are found through this link then
        send Filter Update to filter.neighbor;
    end
end
```

The Filter Update and Sequence Update messages are represented in algorithms 6 and 7 for the FIFO ordering configuration.

Algorithm 4: Publication Advertisement

```
add publisher and topic to last hop filter;
if there is interest in topic then
    foreach filter in neighbor filter set do
        foreach subscriber in filter.subscribers do
            if filter.address != lastHopAddress then
                send Filter Update to filter.neighbor;
            end
        end
    end
end
```

Algorithm 5: Unsubscription Advertisement

```
remove topic from last hop filter;
foreach filter in neighbor filter set do
    if publisher is found through filter and (filter is not
     for parent or last hop) then
        send Advertise Unsubscription to
          filter.neighbor;
    end
end
```

Algorithm 6: Filter Update

```
if new topic being added to last hop filter then
    foreach publisher in known publishers do
        if publisher publishes topic and there is no
          interest for this publisher in last hop filter
          then
            send Sequence Update message to last
              hop;
        end
    end
end
foreach filter in neighbor filter set do
    if publisher for topic is found through filter and
     filter is not to last hop or parent then
        send Filter Update message to filter.neighbor;
    end
end
```

Algorithm 7: Sequence Update

```
update local sequence number for publisher;
foreach filter in neighbor filter set do
    foreach topic in publisher topics do
        Ifthere is interest in topic from filter send
          Sequence Update to filter.neighbor;
    end
end
```

4. Tests

All tests were performed in a single machine. To facilitate the conduction of tests, a process called "Puppet Master" was developed to parse configuration files, that define the overlay-network topology as well as participant processes (Broker, Publisher and Subscriber) for each site, and also simple scripts to control the participants in the system.

We chose a configuration, defining a simple topology, with which to test the system's performance, using the various configuration options described earlier.

The tests incided mainly on the filter routing policy and FIFO ordering, focused on analyzing the routing performance.

Though the delivered SESDAD application still had some problems (sometimes it deadlocks), we try to analyze why this happended and what was the cause in the following sections.

4.1. Configurations

The main concern is the filtering FIFO configuration. Extensive testing was made to ensure coherency in sequence numbers while also avoiding deadlocks.

Configuration file 1 is an example of configuration that was used to define the topology and participant processes in the test analyzed in the next section.

Configuration file 1: Filter-FIFO-test.

```
Site site0 Parent none
Site site1 Parent site0
Site site2 Parent site0
Site site3 Parent site2
Site site4 Parent site3
Site site5 Parent site3
Process broker0 is broker On site0 URL tcp://localhost:3330/broker
Process subscriber0 is subscriber On site0 URL tcp://localhost:1110/sub
Process broker1 is broker On site1 URL tcp://localhost:3331/broker
Process subscriber1 is subscriber On site1 URL tcp://localhost:1111/sub
Process broker2 is broker On site2 URL tcp://localhost:3332/broker
Process broker3 is broker On site3 URL tcp://localhost:3333/broker
Process subscriber3 is subscriber On site3 URL tcp://localhost:1113/sub
Process broker4 is broker On site4 URL tcp://localhost:3334/broker
Process subscriber4 is subscriber On site4 URL tcp://localhost:1114/sub
Process broker5 is broker On site5 URL tcp://localhost:3335/broker
Process subscriber5 is subscriber On site5 URL tcp://localhost:1115/sub
Process publisher00 is publisher On site5 URL tcp://localhost:2220/pub
```

Output Log excerpts 1, 2 resulted from this configuration.

A slight modification to this configuration was made to demonstrate a limitation of the delivered SESDAD application. By introducing a new publisher, in some situations, the system deadlocks. Thus, in the Limitations section we analyze this problem and try to come up with a solution.

4.2. Result analysis

We first analyze the routing mechanism by having just a sub-portion of the tree subscribe to a publisher's topic. And then perform unsubscriptions to verify the correct behaviour of the filtering updates. The script used is represented in Script file 1.

Script file 1: Routing Test.

```
Subscriber subscriber3 Subscribe /p00-0
Subscriber subscriber4 Subscribe /p00-0
Subscriber subscriber5 Subscribe /p00-0
Publisher publisher00 Publish 180 Ontopic /p00-0 Interval 200
Publisher publisher00 Publish 180 Ontopic /p00-1 Interval 200
Wait 15000
Subscriber subscriber3 Unsubscribe /p00-0
Wait 10000
Subscriber subscriber4 Unsubscribe /p00-0
Subscriber subscriber5 Unsubscribe /p00-0
Wait 10000
Subscriber subscriber0 Subscribe /p00-0
Subscriber subscriber1 Subscribe /p00-0
Wait 5000
Subscriber subscriber5 Subscribe /p00-1
Status
```

The test performed as expected, although there is definitely a delay in initial setup on Advertisements. After this initial phase the system performs quite well while only routing messages to the required brokers.

Shortened output logs are presented in order to demonstrate the correct functioning of the system in these conditions.

Log excerpt 1: Intital 3 subscriber state log.

```
[12:10:23] Asking publisher 'publisher00' to publish on topic '/p00-0' !
[12:10:24] Asking publisher 'publisher00' to publish on topic '/p00-1' !
[12:10:25] [subscriber3] Broker confirmed subscription of topic '/p00-0'
[12:10:25] [subscriber4] Broker confirmed subscription of topic '/p00-0'
[12:10:26] [subscriber5] Broker confirmed subscription of topic '/p00-0'
[12:10:27] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 0|| Sequence: 1
[12:10:27] [Routing - 'broker3']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 0|| Sequence: 1
[12:10:28] [Subscriber - subscriber3] Topic: /p00-0||Message: 0 || SEQ: 1
[12:10:28] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-1|| Message: 1|| Sequence: 2
[12:10:28] [Routing - 'broker3']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-1|| Message: 1|| Sequence: 2
[12:10:28] [Routing - 'broker4']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 0|| Sequence: 1
[12:10:28] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 2|| Sequence: 3
[12:10:29] [Subscriber - subscriber4] Topic: /p00-0||Message: 0 || SEQ: 1
[12:10:29] [Routing - 'broker4']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-1|| Message: 1|| Sequence: 2
[12:10:29] [Routing - 'broker3']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 2|| Sequence: 3
[12:10:29] [Subscriber - subscriber3] Topic: /p00-0||Message: 2 || SEQ: 3
[12:10:29] [Routing - 'broker4']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 2|| Sequence: 3
[12:10:29] [Subscriber - subscriber4] Topic: /p00-0||Message: 2 || SEQ: 3
[12:10:30] [Subscriber - subscriber5] Topic: /p00-0||Message: 2 || SEQ: 3
[12:10:30] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-1|| Message: 3|| Sequence: 4
[12:10:30] [Routing - 'broker3']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-1|| Message: 3|| Sequence: 4
[12:10:30] [Routing - 'broker4']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-1|| Message: 3|| Sequence: 4
[12:10:30] [Subscriber - subscriber3] Topic: /p00-0||Message: 4 || SEQ: 5
[12:10:30] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
       Topic: /p00-0|| Message: 4|| Sequence: 5
[12:10:30] [Subscriber - subscriber5] Topic: /p00-0||Message: 4 || SEQ: 5
```

As can be observed on output Log excerpt 1 the system starts routing messages to interested subscribers and only brokers in the path to said subscribers receive the messages.

Note that subscribers do not receive the messages from topic /p00-1, though brokers do receive these messages, this is to avoid adding a further mechanism on top to update the sequence numbers after the initial advertisement phase. Instead the system sends all publisher messages if any of the topics for that publisher have to be sent to an interested subscriber. This way the brokers know the sequence numbers required for delivering the messages in the correct order.

When dealing with unsubscriptions the system updates the filters, and messages are no longer spread through the system, stopping at "broker5", as shown in output Log excerpt 2.

Log excerpt 2: No subscriber state log.

```
[12:10:49] [subscriber4] Broker confirmed unsubscription of topic '/p00-0'
[12:10:49] [subscriber5] Broker confirmed unsubscription of topic '/p00-0'
[12:10:49] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-1|| Message: 189|| Sequence: 190
[12:10:49] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-0|| Message: 190|| Sequence: 191
[12:10:49] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-1|| Message: 191|| Sequence: 192
[12:10:49] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-0|| Message: 192|| Sequence: 193
[12:10:49] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-1|| Message: 193|| Sequence: 194
[12:10:49] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-0|| Message: 194|| Sequence: 195
[12:10:50] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-1|| Message: 195|| Sequence: 196
```

4.3. Limitations

During high throughput tests (publishing interval bellow 200ms) the systems crashes, probably because the tests are being conducted in a single machine.

It is possible that such a problem arises from too many messages with need for processing. Since the system is threaded it is likely that message delivery to subscribers never gets a chance to run, thus the message wait queue gets bigger and bigger until each message processing cycle (threads performing delivery function) takes too long and connections are never terminated, leading to the .net library issuing a refuse connection message.

Another problem arises when a new publisher is introduced on "site3" using Configuration file 1. In this situation the system deadlocks as can be observed in Log excerpt 3.

Log excerpt 3: Two publisher deadlock.

```
[12:10:49] [subscriber4] Broker confirmed unsubscription of topic '/p00-0'
[13:31:46] [subscriber3] Broker confirmed subscription of topic '/p00-0'
[13:31:46] [subscriber4] Broker confirmed subscription of topic '/p00-0'
[13:31:47] [subscriber5] Broker confirmed subscription of topic '/p00-0'
[13:31:48] [Routing - 'broker5']Message: From: 'tcp://localhost:2220/pub' ||
      Topic: /p00-0|| Message: 0|| Sequence: 1
[13:31:48] [Routing - 'broker3']Message: From: 'tcp://localhost:2221/pub' ||
      Topic: /p01-1|| Message: 0|| Sequence: 1
[13:31:48] [subscriber5] Broker confirmed subscription of topic '/p01-1'
[13:32:1] ASking subscriber subscriber3 to unsubscribe topic /p00-0
[13:32:11] ASking subscriber subscriber4 to unsubscribe topic /p00-0
```

Further detailed analysis of the log output of participant processes suggests the problem lies with the locks during Publication Advertisements. Although we could not find a solution for the problem, it migh be a simple bug in the code.

The total-ordering guarantee could have been improved, even in the flood routing policy. The messages could have been sent to the root broker "asynchronously" and then flooded, or filtered, through the whole overlay network starting at the sequencer.

The initial solution for filtered total-ordering was not very efficient, requiring each message to not only go through the sequencing phase but also the advertisement phase, although this implementation was never developed.

Unfortunately the system is not fault-tolerant, meaning any crashes in any of the systems processes means a total system failure. It also fails to deliver the filter routing policy in conjuntion with total-ordering, although total-order works fine with a flooding policy.

Some functions of the "Puppet Master" testing application were not fully developed. The "Crash" command causes the whole system to crash, since no replication was developed, and the "Freeze" command might not work in some situations.

5. Future Work

The delivered SESDAD application leaves room for many improvements, ranging from fault-tolerance, by implementing replication, to guarantees regarding total-order with a filter routing policy.

It also lacks the necessary code to fully deploy the system in a distributed fashion, in fact the system expects a "PuppetMaster" to exist and will not function if one is not present.

Despite this, the implementation of a fully deployable version requires only small changes to way the configurations are processed. There would be a need to have separate configuration files for each participant process, or at least a configuration file for each site.

Since the system was not tested in a distributed environment, future work could focus on testing the adopted solutions in such an environment to determine the system's deployability in a distributed environment.

6. Conclusion

The proposed system delivers most of the functionalities requested except for the fault-tolerance.

Despite some of the limitations, the system performs according to expectations regarding the routing policies and ordering guarantees.