Fundamental Research Issues in Performance Analysis of Runtime Status Collection and Aggregation

Stefan Witt, WSU ID 10539995
stefan_witt@wsu.edu

EE 595 – Directed Study, Summer 2002
Instructor: David Bakken

1 Introduction

This report presents the results of a multicast performance analysis. Performance in this context means the throughput of the multicast system, which depends not only on the available network bandwidth, but also on the speed of the multicast group members. These parameters may change over time, because the bandwidth can vary, and the workload of the group members may vary as well. Thus, the throughput is time-dependent. To perform run-time analysis of a multicast system, a data fusion system is used in this project.

"Mr. Fusion" is a data fusion middleware framework [AMB02]. It is used to collect data from multiple client replicas, and categorize these data as correct or incorrect according to a set of well-defined rules. The incorrect data are then stored as erroneous values in a multidimensional hierarchical database, a data warehouse. Data fusion is performed by aggregating the collected data. Queries on the data can be answered by the data warehouse.

Mr. Fusion consists of two basic components, the Fusion Core, and the Fusion Status Service [AMB02]. While the former component is responsible for collecting data from the replicas and categorizing them, the latter is the data warehouse that aggregates the collected values and answers the queries. The Fusion Status Service is presented in [M02].

In this project Mr. Fusion will be used to detect slowdown failures of subscribers to an atomic multicast. In [BHO+99] it has been shown that the multicast throughput dramatically decreases if even one client suffers from a slowdown failure. This is due to the fact that ACKs are sent late by the slow receiver, and the multicast system as a whole slows down because of that.

In order to be able to detect slowdown failures, the Fusion Status Service needs information about the response times of the clients. This can be inferred from ACKs that are sent by the clients or by ballots of one data fusion session. Then the Fusion Status Service can be used to query for slow clients by an appropriate query.

This report is organized as follows: Section 2 introduces the multicast system that is used and discusses its properties. Section 3 presents the software architecture of the multicast performance analyzer, and section 4 demonstrates the system in a sample session. Finally, section 5 shows the results of the project.

2 The Multicast Service

The multicast system that is used in this project is GroupCom developed by K. Dorow. It provides a totally ordered atomic multicast service[1]. These properties are defined as follows (see [VR01]): A multicast system has *total ordering* if any two messages delivered to any pair of participants are delivered in the same order to both participants. It is *total* if it is atomic and reliable, i.e. all messages are received by all members in exactly the same order.

Total order can be achieved by different algorithms (see [VR01]), for example by using a uniform consensus algorithm (total ordering and uniform consensus are equivalent) or by a sequencer algorithm. The main fact is that each receiver of a message has to send an acknowledgement message (ACK) back to the sender. This ACK is fundamentally important in this project, because it is used to measure the time that the receiver needs to process the incoming message. In an atomic multicast the sender needs to wait in order to collect all ACKs from the receivers before it proceeds, thus the performance of the multicast system depends on the reply times of the receivers. In [BHO+99] it has been shown that the multicast performance slows down if even one of the group members suffers from a slowdown failure, i.e. if one of the receivers needs more time to answer the incoming messages than the other receivers.

GroupCom is written in Java, and it uses a lot of parallel threads – about 200 – in its implementation. The consequence of this design choice is that it is very slow on certain machines. From JDK Version 1.3 on each Java thread runs as a native process in the Operating System. This is especially slow when running under Linux, because Linux does not have a light weight thread model. As a result, each thread the multicast system spawns is a complete process of the Operating System. This does not happen under Solaris or Windows, which have a different process model, and thus GroupCom is much faster on the latter Operating Systems. The slow behavior can only be compensated partially by using faster computers.

3 Architecture of The Multicast Performance Analyzer

The global architecture of the multicast performance analyzer system is shown in figure 1. It consists of one sender, n receivers, the Multicast Performance Monitor, and the Fusion Status Service.

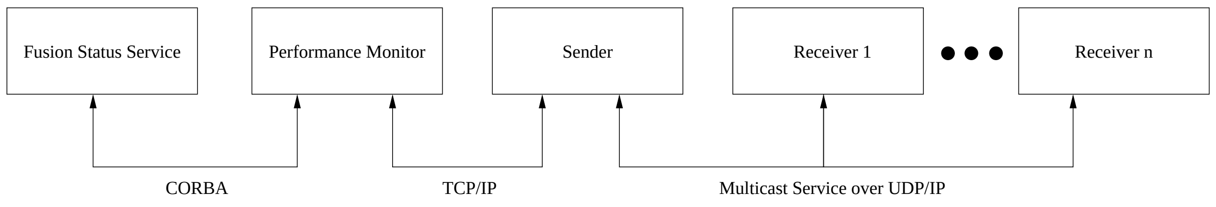

Figure 1: Schema of the global architecture of the Multicast Performance Analyzer.

The way all these programs interact is the following: The multicast sender sends data to the n receivers in the form of messages. The properties of uniformity and total ordering of this multicast system are described in section 2. The receiver programs receive the messages, and they deliver them to the application level with respect to the correct ordering. They respond to each message with an ACK to indicate that they received it correctly. Therefore, the sender is able to measure the time that each receiver requires to send the ACK back. Depending on the system load of the receiver this can take

[1]It is currently still under development, but most parts are already working.

more or less time.

Once the sender has collected all ACK replies from the multicast receivers, it sends the information how long each receiver needed to the performance analyzer.

The multicast performance analyzer is the central part, because it is inbetween the multicast sender and the Fusion Status Service. The task of the performance analyzer is to evaluate the time information that it obtains from the sender and to inform the sender about slow senders. For the evaluation it uses the Fusion Status Service for 3 main reasons. First, the Fusion Status Service provides data fusion, i.e. it automatically aggregates the time information. Furthermore, it provides the possibility to use different criteria for evaluating the aggregated data. Different queries can be defined to analyze the data. And finally, it is possible to subscribe to events, and whenever such a user-defined event occurs, the subscriber is informed. As a simple query to ask for slow multicast members is to check if there is a receiver that took more than twice the average of all reply times. The Fusion Status Service sends the aggregated reply times back to the performance analyzer.

The next step is that the multicast analyzer informs the sender about slow receivers. This is done via an event mechanism, and the sender that subscribed to this event removes the slowest receiver from the multicast group. When the event of a slowdown failure is fired, the performance analyzer does not know, which of the clients suffers from a slowdown failure. Therefore, it queries the Fusion Status Service for all response times of all clients, and using that information, it removes the slowest receiver from the group.

The actual implementation of the whole multicast performance analyzer system consists not only of the multicast performance monitor, but also of a sender and a receiver. They use the GroupCom multicast service to transfer a file, and they interface to the performance monitor. An abstraction layer is used between the sender/receiver and the GroupCom multicast service so that another multicast service could be used instead of GroupCom.

To simulate a slowdown failure the multicast receiver uses two delays: First, each message is delayed by a specified time, and second, the receiver goes to sleep for a specified percentage of its CPU time.

4 Sample Session

The distributed system that is used in this sample session is shown in figure 2. For this session the computers of the research labs at Washington State University are used. The directories that contain the files for this project and the Fusion Status Service need to be installed to an arbitrary place that can be accessed by all of the machines. The configuration is completely done in the `Makefile`, i.e. the directory paths and the names of the machines is set up there. The only exception is the file `db_properties`, which is in the Fusion Status Service directory. The settings are self-explanatory, and they are already set for this sample session. It is important that the Fusion Status Service and the performance monitor need the Java Development Kit 1.4, because they use the CORBA and XML features that are new in this JDK release.

The sequence of shell commands that need to be issued are[2]:

1. Start the Fusion Status Service:
```
sands-l2% cd PerformanceAnalysis
```

[2]Note that the paths may vary if local logins are used. This example assumes remote logins (using `slogin`, for example) to all machines.

```
sands-l2% setenv PATH $PATH/local/j2sdk1.4.0/bin:
sands-l2% setenv DISPLAY some-X-server:0.0
sands-l2% make db
sands-l2% make startsvrgui
```
Now click on “Start Java ORBD” and “Start Server.”

2. Start the Multicast Performance Monitor:
```
sands-l2% cd PerformanceAnalysis
sands-l2% make runmonitor
```

3. Start the receivers:
```
nif-c6% cd PerformanceAnalysis
nif-c6% make runreceiver1

nif-c7% cd PerformanceAnalysis
nif-c7% make runreceiver2

nif-c21% cd PerformanceAnalysis
nif-c21% make runreceiver3
```

4. Start the sender:
```
sands-l1% cd PerformanceAnalysis
sands-l1% make runsender
```

All these computers run Linux. The machines `sands-l1` and `sands-l2` are running with 1.5 GHz, and therefore they are used for the heavy-loaded programs, whereas the receivers run on slower machines with 600 MHz. In this example, receiver 1 is delayed 500ms (50%), receiver 2 is delayed 10ms (1%), and receiver 3 is delayed 100ms (10%). Thus, receiver 1 is the slowest one, but it will not be removed from the group, because the sender process is done too fast to notice it. When changing the delay of the sender in the program file `PerformanceAnalysis/Sender.java` from `Thread.sleep(200);` to `Thread.sleep(1200);`, then the sender notices the slow member and removes it from the group.

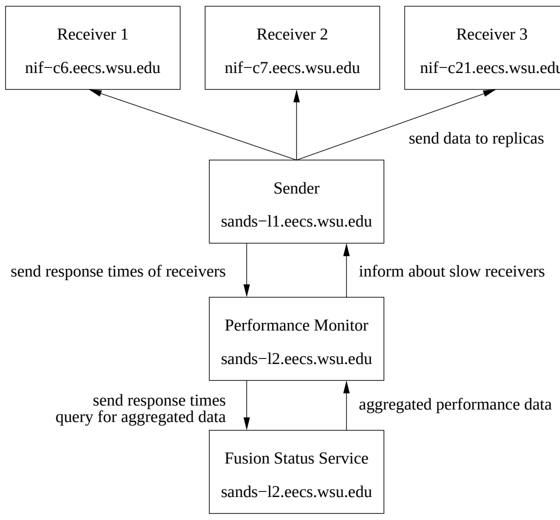

Figure 2: Setting of the sample session. The boxes depict the computers, and the text in them describes their role and name. The arrows indicate the information flow.

In this example, the standard criteria is used to detect a slow group member. It considers a group member failed if its response time is more than twice the average response time of all receivers. The XML code is written in the file `Performanceanalysis/query.xml`, and it is as follows (for an explanation see [M02]):

```
<?xml version='1.0' encoding='utf-8'?>
<expr>
  <query type="time">
    <queryReq unit="allReplicas" measure="replica"/>
    <queryReq unit="time" measure="reason"/>
    <queryReq unit="allSessions" measure="all"/>
  </query>
  <thereIsGt/>
  <expr>
    <query type="time">
      <queryReq unit="allReplicas" measure="all"/>
      <queryReq unit="time" measure="reason"/>
      <queryReq unit="allSessions" measure="all"/>
    </query>
    <mult/>
    <int value="2"/>
  </expr>
</expr>
```

For the Multicast Performance Monitor to obtain information about the response times of all receivers, the query from the file `PerformanceAnalysis/query-allreplicas.xml` is used (for an explanation of the XML code see [M02]):

```
<?xml version='1.0' encoding='utf-8'?>
<expr>
  <query type="xml">
    <queryReq unit="allReplicas" measure="replica"/>
    <queryReq unit="time" measure="reason"/>
    <queryReq unit="allSessions" measure="all"/>
  </query>
</expr>
```

5 Results

The goal of this project was to analyze the multicast performance when at least one receiver suffers from a slowdown failure. This result is yet to come, because the GroupCom multcast system currently supports unordered and FIFO (first-in first-out) ordering only. So far it can be observed that the multicast sender carries on sending despite of the slow member. The slow member is ignored and after a timeout it is assumed to be failed. This behaviour would not happen if totally ordered atomic multicast was used, because the sender would wait for each receiver to make sure that the message was delivered; it would ensure reliable delivery.

Another limitation that this project is suffering from is the slow performance of Java when multi-threaded programs are used. The sender must slow down to a maximum of 10 messages per second (each 7 KB long) in order not to be too fast for the receivers.

Apart from these limitations the multicast performance analyzer works. The main result of the project is that the Fusion Status Service can be used to detect slow members of the multicast receivers. This fulfills the expectations, because the Fusion Status Service was designed for aggregating data at runtime.

References

[BHO+99] K. P. Birham, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu: *Bimodal Multicast*, ACM Transactions on Computer Systems, Vol. 17, No. 2, May 1999, pp. 41–88
Available at `http://www.cs.cornell.edu/ken/bimodal.pdf`.

[AMB02] A. Franz, R. Mista, D. Bakken, C. Dyreson, M. Medidi: *Mr. Fusion: A Programmable Data Fusion Middleware Subsystem with a Tunable Statistical Profiling Service*, to appear in Proceedings of the International Conference on Dependable Systems and Networks (DSN-2002), IEEE/IFIP, June 23–26 2002, Washington, DC
Available at `http://www.eecs.wsu.edu/~bakken/MrFusion-DSN02.pdf`.

[M02] R.Miśta: *Fusion Status Service: Data Warehousing And Temporal Queries of Runtime Status Data*, MS Thesis, Washington State University, 2002

[VR01] P. Verı́ssimo, L. Rodriguez: *Distributed Systems For System Architects*, Kluwer Academic Publishers, Boston 2001