Increasing the availability of a service through Hot Passive Replication

by

John Bengtson and Ola Jigin

LIU-IDA/LITH-EX-G–15/049–SE

2015-06-20
Final thesis

Increasing the availability of a service through Hot Passive Replication

by

John Bengtson and Ola Jigin

LIU-IDA/LITH-EX-G–15/049–SE

2015-06-20

Supervisor: Simin Nadjm-Tehrani
Examiner: Nahid Shahmehri
Students in the 5 year Information Technology program complete a semester-long software development project during their sixth semester (third year). The project is completed in mid-sized groups, and the students implement a mobile application intended to be used in a multi-actor setting, currently a search and rescue scenario. In parallel they study several topics relevant to the technical and ethical considerations in the project. The project culminates by demonstrating a working product and a written report documenting the results of the practical development process including requirements elicitation. During the final stage of the semester, students form small groups and specialise in one topic, resulting in a bachelor thesis. The current report represents the results obtained during this specialization work. Hence, the thesis should be viewed as part of a larger body of work required to pass the semester, including the conditions and requirements for a bachelor thesis.
Abstract

This bachelor thesis examines how redundancy is used to tolerate a process crash fault on a server in a system developed for emergency situations. The goal is to increase the availability of the service the system delivers. The redundant solution uses hot passive replication with one primary replica manager and one backup replica manager. With this approach, code for updating the backup, code for establishing a new primary and code to implement fault detection to detect a process crash has been written. After implementing the redundancy, the redundant solution has been evaluated. The first part of the evaluation showed that the redundant solution can deliver a service in case of a process crash on the primary replica manager. The second part of the evaluation showed that the average response time for an upload request and a download request had increased by 31% compared to the non-redundant solution. The standard deviation was calculated for the response times and it showed that the response time of an upload request could be higher compared to the average response time. This large deviation has been investigated and the conclusion was that the database insertion was the reason.
Contents

1 Introduction
   1.1 Purpose .................................. 2
   1.2 Problem statement ...................... 2
   1.3 Approach ................................ 2
   1.4 Delimitations ............................ 2

2 Background
   2.1 Hardware and software in the reference system .... 3
   2.2 Hardware and software redundancy .............. 3
   2.3 Replication ................................ 3
      2.3.1 Active replication .................... 3
      2.3.2 Passive replication ................... 4
   2.4 Linux command line tools .................... 5
      2.4.1 Pgrep ................................ 5
      2.4.2 Ps ................................... 5

3 Solution Space
   3.1 Backup replica manager ........................ 6
   3.2 Updating the backup ........................ 6
   3.3 Fault detection ............................ 7
   3.4 Establish a primary ......................... 9
   3.5 Architecture ................................ 10

4 Evaluation
   4.1 Experiment setup ........................... 12
   4.2 Testing for fault tolerance .................. 12
      4.2.1 Normal operation scenario ............. 12
      4.2.2 Crash detection ........................ 12
      4.2.3 Failover after crash .................... 13
      4.2.4 Crash scenarios ......................... 13
   4.3 Measuring performance ....................... 14
      4.3.1 Normal operation cases ................ 15
      4.3.2 Failover case .......................... 15
      4.3.3 Metrics ............................... 16
      4.3.4 Results ............................... 16
      4.3.5 Observations ........................... 18
   4.4 Deviation measurements ...................... 18
      4.4.1 Result ............................... 18
      4.4.2 Discussion ............................ 19
   4.5 Lessons learnt ............................ 20

5 Conclusion .................................. 21
   5.1 Future work .............................. 21
1 Introduction

This thesis is done in parallel with a project where the authors have developed a critical communication service. The system is meant to be used in emergency scenarios, such as natural disasters. Therefore it is important that the service is available at all times.

The system being analyzed in this thesis uses a client-server architecture. To deliver an available service and make the system fault-tolerant, redundancy is necessary \[1\]. We will look at a fault that can affect the availability and add redundancy to tolerate this fault.

![Figure 1: The reference system with hardware components and processes running on them](image)

The system described in figure 1 consists of a client side and a server side. The client side uses a mobile handset with an application that has a contact list and a map. In the contact list, a user can look up and add contacts. The map is used for accidents, a user is able to add accidents on the map and look up other accidents.

The application has a request handler that performs requests to the server. There are two types of requests, download and upload. The data in these requests are mission related, which could be information about an accident or a contact. Upload requests are used when the client wants to insert information about a contact or a map event into a database. Download requests are used when the client wants to retrieve information about a contact or a map event from a database. The request handler communicates
with a web server on the server side, which is responsible for processing every client request.

1.1 Purpose

The purpose of this thesis is to add redundancy to improve the availability of the system described in figure 1 in case of a process crash. Measurements will be made to analyze the difference in performance between the reference system and the redundant system.

1.2 Problem statement

In figure 1, we exemplify a fault. This fault is a process crash on the server. The consequence of the fault is that the service will be unavailable, which can be classified as a service halt failure. This means that no service is delivered from the server side. Since the service is used in emergency situations, the fault must not lead to a failure and redundancy has to be added.

To prevent a service failure the following questions must be answered:

- How do we implement a redundant solution to tolerate a process crash on the server?
- How will the performance of the system be affected by the selected solution?

1.3 Approach

This thesis consists of two parts. First we will determine what kind of redundant solution is needed to tolerate a process crash on the server and implement it. Afterwards we will study whether if the solution works as intended and measure the response time for the reference system and the redundant solution to evaluate the difference in performance. The redundant solution must satisfy the following requirements:

- In case of a process crash, the service must still be available to the client.
- The average response time for a client’s request should at most be twice as long compared to the reference system.
- The added redundancy should not change the basic functionality of the system, such as contact insertions and extractions.

1.4 Delimitations

Due to time limitations for this thesis, we do not consider other faults than a process crash on the server side within our fault model.
2 Background

Before adding redundancy to the reference system, we need to study what kind of redundancy is needed and how to implement it. This chapter will give the reader the necessary information to understand the choice of solution. At first we describe the hardware and software that is used. Secondly we describe what kind of redundancy that is relevant and what solutions that are possible. Finally we describe tools used in the selected solution.

2.1 Hardware and software in the reference system

The operating system on the server is a Linux distribution called Debian\(^1\). The server is running an Apache\(^2\) process, which makes it a web server. The code on server side is written in PHP\(^3\). The data is stored on a MySQL\(^4\) database located at another server. The application on the client side is written in Java on the Android platform. The client communicates with the web server using the Hypertext Transfer Protocol (HTTP) over Transmission Control Protocol (TCP).

2.2 Hardware and software redundancy

Redundancy is the use of extra resources that are not needed when everything works as intended, but are necessary when components in a system fail. If a process crash occurs within the reference system, the system must be able to deliver service to the client. To replace a crashed process, software redundancy can be added\(^3\). In addition, hardware redundancy can also be added to host the redundant software\(^3\). This type of redundancy is referred to as replication.

2.3 Replication

Replication means creating several copies (or replicas) of data and files. In that way, if some copies are unavailable, there are other copies to use. Schemes for managing replicas can be divided into two major types, active replication and passive replication\(^4\).

2.3.1 Active replication

In active replication there are several replica managers responsible to process each request sent by the clients. Each replica manager updates its state and returns a response. The clients multicast the requests to every replica manager. If a replica manager crashes it is transparent to the client because there are still other replica managers functioning\(^5\).

\(^1\)https://www.debian.org/
\(^2\)https://httpd.apache.org/docs/2.2/
\(^3\)http://php.net/
\(^4\)http://www.mysql.com/
2.3.2 Passive replication

Passive replication is usually called the primary-backup approach. It consists of a primary replica manager and at least one backup replica manager. The primary is responsible for the communication with clients, which involves processing the requests and returning a response. If the primary crashes, it has to be replaced by a backup [6].

The primary also makes sure that each backup updates its state according to the client’s request in a given interval, or logs all requests, which the backup uses to establish a new primary in case of a primary crash. Cold passive replication and warm passive replication are two different types of passive replication. The difference between these two is how often the state of the backup’s replica is updated [7].

In cold passive replication, the primary never updates the backups during its uptime. Instead, the primary logs all requests that have been processed. When a crash occurs on the primary, the log is used for recovery by the backup, which updates its replica before acting as the new primary. If a large amount of requests have been processed, the time for the failover procedure can be long.

In warm passive replication, the primary updates the backups at certain intervals. In between two updates, the primary logs all the requests. A variant of warm passive replication is hot passive replication. With hot passive replication, updates occur after the primary has processed the request and before the primary sends a response to the client. With this approach, the primary keeps no log of requests that have been processed. The response time depends on what type of hot passive replication that is used. There are five sub-classes of hot passive replication [7].

- Blocking on processing - all answer
- Blocking on processing - first answer
- Blocking on delivery - all answer
- Blocking on delivery - first answer
- Non-blocking

Blocking on processing means that the primary does not send a response to the client until the backup has processed the request. Blocking on delivery means that the primary does not send a response to the client until the request has been delivered to the backup. Non-blocking means that the primary sends a response immediately when it has computed the response. All answer means that it waits until all backups have answered and first answer means that it just waits for the first backup to answer [7].
2.4 Linux command line tools

This section briefly describes the Linux command line tools used in this thesis.

2.4.1 Pgrep

Pgrep is a Linux command line tool that lists the running processes identification numbers based on a given selection criteria. The criteria can be the name of the process and the owner of the process.

2.4.2 Ps

Ps is a Linux command tool used to report the state of running processes. It takes a snapshot of the status. Ps can be given a selection criteria to return the state of a specific process. The state returned by ps can differ depending on the state of the process. The following state codes can be returned:

- D - Uninterruptible sleep
- R - Running or runnable
- S - Interruptible sleep
- T - Stopped
- X - Dead
- Z - Defunct ("zombie") process

---

5http://linux.die.net/man/1/pgrep
6http://linux.die.net/man/1/ps
3 Solution Space

In this chapter we describe the choices between different options and the implemented solution.

As mentioned in section 2.3 there are two major replication techniques that can be used, passive or active replication. The differences between these two is that when active is used, a client has to multicast the request to every replica manager, regardless the type of the request. In this approach resources are therefore wasted when a download request is sent. In passive replication, only the primary processes a download request, which does not waste resources [6, 8]. We have chosen to add redundancy in form of passive replication to the system.

With the use of passive replication, the following has to be added:

1. Additional hardware and software in form of a backup.
2. Code for updating the backup according to the primary’s state.
3. Code to implement fault detection to detect a process crash.

3.1 Backup replica manager

To avoid service failure in case of a process crash, another process has to be ready to take over. An additional process is therefore necessary. One additional process is enough since the scope for this thesis is to tolerate a single process crash. As described in section 2.2 we have chosen to place the process on an additional hardware. With the use of additional hardware, the fault model could have been extended to include processor crash as well. Due to time limitations, we did not extend the fault model. The additional process that has to be added is an Apache process. It will follow the same setup as in section 2.1. This results in having one backup replica manager.

3.2 Updating the backup

To decide what type of passive replication was most suitable for the reference system we had to consider in what situations it was going to be used in. Since the system is supposed to be used in emergency situations, availability is of most importance. The choice stood between cold or warm passive replication. If a process crash occurred and cold passive replication was used, the failover time would increase and affect the availability much more than if warm passive replication would be used [9]. Although, warm passive replication would reduce the performance for a single request more than cold passive, but the response time should still be acceptable [7]. We choose
to implement hot passive replication, which means that the backup will receive every upload request sent by the client, directly after the primary has received it. In that way, the availability is increased since the backup is up to date just a short while after each request is sent. This means that the backup will be ready to replace the primary directly after the primary has crashed.

With the hot passive approach the primary will update the backup at the end of each request as described in section 2.3.2. The question was what type of blocking should be used. No blocking was not an option since there is no guarantee that the backup has received the update. If blocking on process was used, the response time would increase, because the execution time at the backup and the network time for reaching the backup would be added. If blocking on delivery was used, the response time would also increase but only the network time would be added. Both blocking on delivery and blocking on process guarantee that the backup has received the request. Since blocking on delivery is faster, we choose to use that implementation [7]. Blocking on delivery is shown in figure 2.

![Figure 2: The update procedure performed by the primary replica manager](image)

3.3 Fault detection

One major task with passive replication is to detect if a process has crashed [10]. In this thesis, once a process has crashed it will not recover without external interaction, it is classified as a permanent fault. The process crash can occur at any time. It is important that the fault detector does not make false detections, it should know the difference between a crashed process
and a slow process. We have chosen to place the fault detector at the same hardware that hosts an Apache process. This means that one fault detector is placed on the primary server and another is placed on the backup server.

The fault detector at the primary performs three tasks. The first task is to listen for incoming messages from a client on a specific port number and respond. It communicates with the client through sockets. This fault detector responds with the address of the server the client should connect to. This procedure is shown in figure 3a. The second task is to check the state of the Apache process on the primary. It uses the Linux command line tools pgrep and ps to retrieve the state of the process. With pgrep it retrieves the process identification number. This identification number is used with ps to retrieve the state of the Apache process. The returned process state code depends on what kind of state the Apache process is in as described in section 2.4.2. If the state code D, R, S or T is returned the process has not crashed. If X or Z is returned the process is classified as crashed. If pgrep does not return any process identification number the process is classified as crashed. If the process crash after the process identification number has been retrieved but before the ps command has been executed, ps will return no state code which is classified as a crashed process. The fault detector checks the process state continuously, which means that the rate of arrival of requests has to be much lower than the rate of checking the state by the fault detector. Otherwise the client will try to reconnect to the crashed web server. This task is shown in figure 3b. The third task is to retrieve the address of the backup web server. This is done by continuously sending messages to the fault detector at the backup. If the Apache process on the backup is alive, it will receive the address of the backup web server. If it has crashed, it will receive a response that no backup is available. This is a type of ”Are you alive?” messages and is shown in figure 3a.

The fault detector at the backup performs two tasks. The first task is to listen for incoming messages from the fault detector at the primary on a specific port number and respond. It communicates with the primary through sockets. This fault detector responds with the address to the backup web server. The second task is to check the state of the Apache process on the backup. This is done in the same way as with the fault detector at the primary with the Linux command line tools pgrep and ps.

These fault detectors were written in Java. The advantages of this implementation is that the client does not need to know the address of every replica manager. Server updates can be done without configuring the clients. The client only communicates with the fault detector at the primary and therefore it only has to know that address. Figure 3 shows how the fault detectors work.
3.4 Establish a primary

The only time a new primary will be established is when the fault detector at the primary has detected an Apache process crash. The establishment can occur during a request or between two requests. If it happens during a request, an IO-exception will be thrown at the client side when it tries to retrieve a response code from the crashed process. If it happens between two requests, an IO-exception will be thrown at the client side when it tries to establish a connection. After an exception has been thrown, the client reconnects to the fault detector to retrieve a web server address. The fault detector hopefully has detected a process crash and will return the address of the backup to the client and the backup will become the new primary.

Once the backup has become the new primary, there are three scenarios that describe at what stage the client’s request was in [5].

1. The primary crashed before updating the backup, and thus before sending a response to the client.
2. The primary crashed after updating the backup, but before sending a response to the client.
3. The primary crashed after sending a response to the client.
In the first case, the backup will not have processed the request and when the client resends its request, the backup will process it as normal. In the second case, the backup has processed it, but the client thinks otherwise. When the client’s request arrives at the backup, the backup needs to discover if it already has processed the request, otherwise it will attempt to write the same data into the database twice. In the third case the client will not have to resend the request.

For case two, we have added a check to prevent duplicate insertions. The backup will check the unique identification of the insertion request, to examine if an entry already exists in the database. If that is the case, the backup will return a response to the client without processing the request again. If not, it will process the request and then return a response. The unique identification for the insertion is sent together with the data from the client. Figure 4 shows the establishment of a new primary.

### Figure 4: The procedure for establishing a new primary during a request

#### 3.5 Architecture

With the replicated solution, we get an extended architecture according to figure 5. The figure shows hardware components and what processes are running on them. With no fault present within the system, the flow of a client’s request is shown in figure 6.
3.5 Architecture

Figure 5: The extended architecture with added redundancy

Figure 6: The procedure of a request when no fault is present
4 Evaluation

In this chapter, we will first test the functioning of the replicated solution. Afterwards we will measure the response time to evaluate the difference in performance between the reference system and the replicated solution.

4.1 Experiment setup

There were two experiment setups. The first setup was for the reference system in figure 1. The second setup was for the replicated solution in figure 5. The mobile handset in both setups used Android 5.0 \[7\] . The servers operating system is the same as in section 2.1. The communication between the mobile handset and the servers was done wireless via Wi-Fi.

4.2 Testing for fault tolerance

To test whether the solution works as intended or not, the following requirements must be fulfilled.

1. Data must be replicated during normal conditions with no fault present.
2. The fault detector should detect whether a process has crashed or not.
3. The backup should not process the same request twice.
4. The client should always switch server if a process crash occurs.

4.2.1 Normal operation scenario

To verify the first requirement, we performed the following test: a client added a contact, which resulted in an upload. Afterwards the client received a response with a successful status code. Both the primary’s and the backup’s databases were examined to make sure that an entry of the contact existed. There was an entry in both databases and the replication was confirmed.

4.2.2 Crash detection

To verify the second requirement, we performed the following test: a client sent several requests in a row. While these requests were being generated, the Apache process was terminated. Before the process was terminated the fault detector returned the address of the primary, but once the process was terminated the fault detector returned the address of the backup. This test was therefore successful and the fault detector was confirmed functional.

\[7\] http://www.android.com/versions/lollipop-5-0/
4.2 Testing for fault tolerance

4.2.3 Failover after crash

To verify the third requirement we performed the following test: a client added a contact which resulted in an upload. The backup acknowledged the update from the primary, and before the primary sent a response to the client, the Apache process was terminated. When the exception was thrown at the client, the client reconnected to the fault detector at the primary to receive the address of the backup. Thereafter the client connected to the backup. The backup checked if it already had processed the request and returned a response to the client. The database was examined to make sure that no duplicated entries existed. This was confirmed and the test was successful.

4.2.4 Crash scenarios

The fourth requirement to verify was that the client always switched server if a process crash occurred at the primary. As shown in figure 7 there are four possible scenarios. The first scenario is if the process crashes between two requests. The second scenario is if the process crashes after the fault detector returned the address of the primary but before the request arrived at the primary. The third scenario is when the process crashes after the primary has received the request, but before updating the backup. The fourth scenario is when the process crashes after the primary has updated the backup but before sending a response to the client.

To verify these four scenarios, the Apache process was terminated at the specific stage of the procedure. In every scenario, both the client and the fault detector behaved as expected. The fault detector returned the backup’s address and the client started to communicate with the backup as the new primary.
4.3 Measuring performance

To evaluate how the performance has been affected by the replicated solution, we have measured the average response time for client requests. As described in section 1.3 one requirement was that the average response time for the replicated solution should at most be twice as long compared to the reference system. We consider that this requirement concerns the scenario with no fault present both within the reference system and the replicated solution. The system is supposed to function properly during its usage period and a process crash should preferably not occur at all. Since our fault model was a single process crash fault, it is therefore of a greater interest to make sure that the response time is acceptable during intervals when no fault is present. During the measurements, the following four cases were considered.

- The first case concerned the reference system. In this case, the system was working as intended with no fault present.
- The second case concerned the replicated solution. In this case, the system was working as intended with no fault present.
- The third case concerned the replicated solution. In this case, the process on the primary had crashed, and the client communicates with the backup as the new primary from the start.
• The fourth case concerned the replicated solution. In this case, a process crash occurred on the primary during a client’s request and a failover was performed.

4.3.1 Normal operation cases

Case one, two and three were tested with tests running both 20 000 download requests and 20 000 upload requests. These requests were sent from one client, and only one request at a time. We performed an equal amount of download and upload requests to get an overall view of the response time and to capture normal problems that may occur on a daily basis, such as network delays. We expect that the average response time will be longer with replication than without for the first three cases.

Since data only is replicated during upload requests, the response time for this type of requests in case two will display the effect of replication. Since no data is replicated in case three, we believe that the average response time will be longer than in case one but shorter than in case two.

With download requests, we believe that there will be no major difference in response time for case two and three. This is because the same type of operation are performed, but on two different servers. Though, case two and three will have a longer response time than case one since the fault detector is used.

4.3.2 Failover case

The fourth case was tested with tests running 20 000 upload requests. These requests were also sent from only one client, and only one request at a time as in section 4.3.1. At each test, a process crash occurred and a failover was performed. As shown in figure 7, a process crash could occur at four stages during the processing of a request. The reason we investigate the failover is to see how fast the backup can replace the primary in case of a process crash. To get an insight of how long the failover could be, we looked at the scenario in which the process crashes just before it was about to send a response to the client. This represent crash scenario four in figure 7. Even if we did not measure the failover for other potential crash scenarios or for download requests, this gives us an overview of how long the failover could be. Other potential crash scenarios could have a shorter or a longer response time, but we believe it would not be much worse than in crash scenario four. We expect the average response time for a failover to be much longer than the average response time for all other cases.
4.3.3 Metrics

For each case, an average response time $\mu$ was calculated using

$$\mu = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i$$

where $N$ is the number of requests and $x_i$ is the response time of each request.

The standard deviation $\sigma$ was also calculated to display the variation in response time using

$$\sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \mu)^2}$$

where $N$ is the number of requests and $x_i$ is the response time of each request.

4.3.4 Results

In this section, the average response time for each case is presented.

Download requests

<table>
<thead>
<tr>
<th>Case</th>
<th>Average response time</th>
<th>Standard deviation</th>
<th>Maximum response time</th>
<th>Minimum response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Reference system no crash</td>
<td>24.3</td>
<td>10.8</td>
<td>357.0</td>
<td>14.0</td>
</tr>
<tr>
<td>2) Replicated system no crash</td>
<td>31.9</td>
<td>17.5</td>
<td>1290.0</td>
<td>20.0</td>
</tr>
<tr>
<td>3) Replicated system with backup as primary</td>
<td>32.1</td>
<td>22.3</td>
<td>1885.0</td>
<td>21.0</td>
</tr>
</tbody>
</table>

Table 1: Average response time, standard deviation, maximum response time and minimum response time for download requests in case one, two and three. Time is displayed in milliseconds.

Table 1 shows the average response time for download requests. The average response time in the second case was 31% higher than the first case and the average response time in the third case was 32% higher than the first case. As expected the average response time with replication was longer than without and the second and third case exhibited similar behavior in terms of response time.
Upload requests

<table>
<thead>
<tr>
<th>Case</th>
<th>Average response time</th>
<th>Standard deviation</th>
<th>Maximum response time</th>
<th>Minimum response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Reference system no crash</td>
<td>33.0</td>
<td>123.4</td>
<td>11256.0</td>
<td>17.0</td>
</tr>
<tr>
<td>2) Replicated system no crash</td>
<td>43.1</td>
<td>98.6</td>
<td>9500.0</td>
<td>24.0</td>
</tr>
<tr>
<td>3) Replicated system with backup as new primary</td>
<td>40.6</td>
<td>88.8</td>
<td>9904.0</td>
<td>23.0</td>
</tr>
</tbody>
</table>

Table 2: Average response time, standard deviation, maximum response time and minimum response time for upload requests in case one, two and three. Time is displayed in milliseconds.

Table 2 shows the average response time for upload requests. The average response time in the second case was 31% higher than in the first case and the average response time in the third case was 23% higher than in the first case. As expected, the response time for case two, with replication of data, was longer than both the reference system in case one and with the backup as the new primary in case three. Also, in case three, the average response time was longer than the reference system in case one, but shorter than the replication in case two. The reason we see an 8% difference between case two and three is because no replication of data occurred in case three, only the fault detector was being used.

These results shows a performance degradation of 31%, which we find satisfying according to the requirement in section 1.3.

Failover

<table>
<thead>
<tr>
<th>Case</th>
<th>Average response time</th>
<th>Standard deviation</th>
<th>Maximum response time</th>
<th>Minimum response time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4) Replicated system failover</td>
<td>679.8</td>
<td>929.0</td>
<td>21294.0</td>
<td>383.0</td>
</tr>
</tbody>
</table>

Table 3: Average response time, standard deviation, maximum response time and minimum response time for upload requests during failover in case four. Time is displayed in milliseconds.

Table 3 shows the average response time for a failover during an upload request. As expected, the average response time was longer than the average response time in all other cases. The response time was 16 times more than in case two. With the use of hot passive replication, we expected the backup to be ready to replace the primary directly after the primary had crashed. Even if the response time was 16 times more than in case two, the backup
was ready. The reason we got a longer response time is because the client had to reconnect to the fault detector and then to the new web server. The backup also had to check if the request already had been processed. We find this result satisfying.

4.3.5 Observations

An interesting observation is the standard deviation for upload requests in all four cases. The maximum response time deviates much from the average response time, as shown in the tables 2 and 3. The reason for this is because there is no upper or lower bound for the response time within the system. The deviations occur both in the reference system and in the redundant system which is an indication that the redundancy is not the cause. However it is of interest to examine the cause of these deviations. Since the deviations occur in a greater extent in the upload requests, we only investigate these. We also only investigate the deviations for case one, two and three and not the failover time in case four due to time limitations.

4.4 Deviation measurements

To measure what part of an upload request that makes the response time deviate, we have measured the time for each part of the overall process. 20000 additional upload requests were generated for the first, second and third case in section 4.3. The measurements consists of the following parts:

\( T_1 \) : The network time between the client and the fault detector. This can only be measured in case two and three.

\( T_2 \) : The execution time for the fault detector. This can only be measured in case two and three.

\( T_3 \) : The execution time at the primary server.

\( T_4 \) : The execution time for the database insertion.

\( T_5 \) : The network time and the execution time for a database insertion.

\( T_6 \) : The network time between the primary and the backup. This can only be measured in case two.

\( T_7 \) : The network time between the client and the primary.

4.4.1 Result

The result shows that the deviations occurred due to network time (\( T_1 \) and \( T_7 \)) or the database execution time (\( T_4 \)). Both these types appeared in case one, two and three. We only looked at response times that exceeded 160 ms, since that is the highest upper bound of the standard deviation in case one, two and three. Each \( T_i \) was measured in milliseconds, except \( T_5 \), it
was measured in seconds. This is because we were not able to measure it in milliseconds due to limitations in the MySQL version used. This means that the measured execution time on the database may have a margin of error of one second. Since the deviations can be as long as 15221 ms it is acceptable to measure the execution time in seconds.

**Network**

Deviations in form of network time could occur due to network problems, for example congestion in the network. These deviations only appeared between the client and the server where the connection was wireless. The deviations were at worst 3841 ms. In case one, 0.23% of the deviations were caused by the network. In case two, 0.57% were caused by the network and in case three, 0.54% were caused by the network.

**Database**

Deviations caused by the execution time on the database were at worst 15221 ms. These deviations occurred in a greater extent than deviations caused by network time, and is the most contributing factor to why the standard deviation is large for every case of the upload requests. Since the average response time was 40.6 ms for case three, a response time at 15221 ms is 375 times higher and affects the average response time and standard deviation overall.

In case one, 0.66% of the deviations were caused by the database. In case two, 0.56% were caused by the database and in case three, 0.88% were caused by the database. In the table below, the worst database deviation for each case is represented.

<table>
<thead>
<tr>
<th>Case</th>
<th>Total time</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
<th>T5</th>
<th>T6</th>
<th>T7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.816</td>
<td>-</td>
<td>-</td>
<td>0.007</td>
<td>12</td>
<td>11.771</td>
<td>-</td>
<td>0.038</td>
</tr>
<tr>
<td>2</td>
<td>10.512</td>
<td>0.09</td>
<td>≈ 0</td>
<td>0.007</td>
<td>10</td>
<td>10.073</td>
<td>0.003</td>
<td>0.339</td>
</tr>
<tr>
<td>3</td>
<td>15.221</td>
<td>0.004</td>
<td>≈ 0</td>
<td>0.01</td>
<td>15</td>
<td>14.998</td>
<td>-</td>
<td>0.209</td>
</tr>
</tbody>
</table>

*Table 4: Deviations for upload requests in case one, two and three. Time is displayed in seconds. - means that no measurement was done for that case and ≈ 0 means that it was less than one millisecond.*

**4.4.2 Discussion**

The conclusion after these measurements is that the deviations caused by database insertions is not from the replicated solution, since they appear for upload requests in the reference system as well. An optimization can be made, but it will affect the structure of the database and thus the basic
functionality. Since a requirement in section 1.3 prevents us from doing this, an optimization was not pursued. On the other hand, deviations caused by the network is a consequence of the replicated solution. Since the replicated solution contains more network connections it will therefore create more deviations.

4.5 Lessons learnt

The measurements do not display how one process crash affects the average response time during a usage period. To investigate this, the use of multiple clients sending requests and the use of buffers at the server side must be considered. Due to time limitations, this was not done, even though the solution supports this. This means that we have not determined how the system behaves in relation to number of clients within the system and the rate of requests.

A parameter that may affect the performance is how often the fault detector at the primary retrieves the web server address of the backup. In this implementation, the fault detector does this continuously, which means more network traffic and more execution time. Instead, this could have been done only when the fault detector at the primary detects an Apache process crash, which means it only would be performed once. Since only one client sending requests have been considered in the measurements, and only sending one request at the time, this have probably not affected the performance. But if multiple clients sending requests at the same time would have been considered, this might contribute to a greater performance degradation.
5 Conclusion

To conclude this thesis we will answer the questions described in section 1.2.

1. **How do we implement a redundant solution to tolerate a process crash on the server?**
   At first we had to decide what type of redundancy should be used. Our conclusion was that replication should be used to tolerate a process crash. To achieve this we had to add several components. The first component that had to be added was another Apache process. The second component was additional hardware to host the Apache process. We also had to add another MySQL database. This resulted in a backup replica manager. Fault detectors were added to detect process crashes and redirect the client to the backup. This resulted in additional software on both the primary and the backup.

2. **How will the performance of the system be affected by the selected solution?**
   The results show that there was a performance degradation. The response time in absence of crashes was 31% higher with the redundant solution than with the reference system. Nevertheless, we are pleased with the result since it satisfies the requirement in section 1.3 that the response time in the redundant solution is lower than twice as long as the response time in the reference system. The performance degradation was expected since the redundant solution contains more network connections and additional code at both the client side and the server side.

5.1 Future work

It would be interesting to investigate how the average response time changes in case of recurrent process crashes. One option is to look at how the response time changes in case of several process crashes during a certain usage period. Further on, multiple clients sending requests could also be considered with the use of buffers at the server side. Both the fault detector and the Apache process has the possibility to buffer incoming requests and process them when resources are available. To implement this based on a suitable buffer size, one has to examine how many clients there will be within the system and what the rate of requests would be.

Another parameter to evaluate is different size of data during upload and download requests. For example, uploading 100 contacts instead of just one.

The replicated solution is fault-tolerant for an Apache process crash fault on the server side. If other faults would be considered, the current implementation would not be fault-tolerant without additional implementation. If a
processor crash on the server side was considered, there is a chance the fault
detectors would not function properly. The client would then not be able
to connect to the web servers. The implementation could be extended, but
then additional redundancy would be needed with a reconfiguration at both
the server side and the client side. Overall, the replication procedure can
be maintained but additional fault detection has to be added. It is possible
to add but probably at the cost of some performance degradation. A new
fault detection is preferable to be able to maintain an acceptable response
time.
Bibliography


Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommerziell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är krävande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© John Bengtson and Ola Jigin