Examensarbete på avancerad nivå

Independent degree project – second cycle

Datateknik 30 hp
Computer Engineering 30 credits

Preventing data loss using rollback-recovery
A proof-of-concept study at Bolagsverket

Max Sjölinder
Abstract

This thesis investigates two alternative approaches, referred to as automatic- and semi-automatic replay, which can be used to prevent data loss due to a certain set of unforeseen events at Bolagsverket, the Swedish Companies Registration Office. The approaches make it possible to recover the correct data from a database that belongs to a stateless distributed system and that contains erroneous- or inaccurate information due to past faults. Both approaches utilize log-based rollback-recovery techniques but make different assumptions regarding the deterministic behaviour of Bolagsverket’s systems. A stateless distributed system logs all received messages during failure-free operation. During recovery, automatic replay recovers the data by enabling the system to re-process the logged messages. In contrast, semi-automatic replay recovers data by utilizing the logged messages to enable officials at Bolagsverket to manually redo lost work in a controlled manner. Proof-of-concept implementations of the two replay approaches are developed on a simplified model that resembles one of Bolagsverket’s electronic services, yet that is general to any stateless system that communicates asynchronously using JMS messages and synchronously using XML sent over HTTP. The theoretical- and performance evaluation was conducted with the aim of producing results general to any system with similar characteristics to those of the model. The results suggest that the failure-free overhead at Bolagsverket is approximately 100 milliseconds per logged message, and that around 3 gigabytes of data must be stored in order to recover one average day’s operation. Further, automatic replay successfully manages to recover one average day’s operation in around 70 minutes. Semi-automatic replay is calculated to require, at a maximum, one workday to recover the same amount of data. It is assessed that automatic replay is a suitable solution for Bolagsverket if it is proven that their systems are fully deterministic. In other cases, it is assessed that semi-automatic replay can be utilized. It is however recommended that further evaluations are conducted before the approaches are implemented in a production environment.

Keywords: Fault tolerance, rollback-recovery, data loss, Bolagsverket.
Acknowledgements

I am very thankful to have had the opportunity to conduct my thesis at Bolagsverket. I would like to thank everyone at Bolagsverket who in any way helped or supported me during this thesis. I would particularly like to express my sincere gratitude to my tutor Martin Bylund, for his excellent support, interesting discussions and continuous encouragement.

I would also like to express my great appreciation to my tutor at Mid Sweden University, Ph.D. candidate Wei Shen, for his wise guidance and constructive feedback.

Finally, I would like to take the opportunity to thank my friends, girlfriend and family for their unconditional and invaluable support during this thesis, as well as throughout my education.
Table of Contents

Abstract ................................................................................................................................. ii
Acknowledgements .......................................................................................................... iv
Terminology ...................................................................................................................... viii

1 Introduction .................................................................................................................. 1
  1.1 Background and problem motivation ...................................................................... 1
  1.2 Overall aim ................................................................................................................ 4
  1.3 Scope .......................................................................................................................... 4
  1.4 Concrete and verifiable goals .................................................................................... 6
  1.5 Outline ....................................................................................................................... 8
  1.6 Contributions ............................................................................................................. 8

2 Theory ............................................................................................................................. 9
  2.1 Distributed systems ................................................................................................... 9
  2.2 Dependable systems .................................................................................................. 10
  2.2.1 Error handling approaches ............................................................................... 11
  2.3 Rollback-recovery in distributed systems ............................................................... 11
  2.3.1 Checkpoint-based rollback-recovery ............................................................... 12
  2.3.2 Log-based rollback-recovery .......................................................................... 14
  2.3.3 Programming transparency ............................................................................. 16
  2.4 Java Enterprise Edition ............................................................................................ 16
  2.4.1 JavaServer Faces ............................................................................................. 17
  2.4.2 Java Message Service ....................................................................................... 17
  2.4.3 Enterprise beans ............................................................................................... 17
  2.4.4 Java Naming and Directory Interface ............................................................. 18
  2.4.5 Java Persistence API ....................................................................................... 18
  2.4.6 RESTful web services with JAX-RS ................................................................. 18
  2.5 Database recovery and restoration ......................................................................... 18
  2.6 Related work ............................................................................................................. 19
  2.6.1 Recovery oriented computing .......................................................................... 19
  2.6.2 Logging and checkpointing tools ...................................................................... 19
  2.6.3 Undo and Redo ................................................................................................. 20
  2.6.4 Related research fields ..................................................................................... 20

3 Analysis of Bolagsverket’s IT systems .......................................................................... 22
  3.1 Overview ................................................................................................................... 22
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

Table of Contents

2013-12-27

3.1.1 Migration from a mainframe architecture ..................23
3.2 Company registration via RFB ..................................24
3.3 Errand handling ....................................................25
3.4 Statistics .............................................................26

4 Methodology ..................................................................28
4.1 Software development methodology ....................................28
4.2 Tools .................................................................28
4.2.1 Hardware ............................................................28
4.2.2 Development tools and software ..................................29
4.2.3 Other software tools .................................................31
4.3 System evaluation ......................................................32
4.3.1 Logging overhead ....................................................32
4.3.2 Data accuracy after replay .........................................33
4.3.3 Recovery time ........................................................33
4.3.4 Storage requirements ...............................................35
4.3.5 Scalability ............................................................35
4.3.6 Implementation complexity .......................................35
4.3.7 Integration complexity .............................................36
4.3.8 Maintainability ........................................................36

5 Implementation ..........................................................37
5.1 Requirements ..........................................................37
5.1.1 E-service model .....................................................37
5.1.2 Proof-of-concept implementation ..................................38
5.2 E-service model ........................................................40
5.3 Proof-of-concept implementation ......................................41
5.3.1 Logging system .......................................................42
5.3.2 Replay system ........................................................44

6 Results ............................................................................53
6.1 E-service model .........................................................53
6.2 Proof-of-concept implementation .......................................54
6.3 Performance test and theoretical evaluation ..........................55
6.3.1 Logging overhead .....................................................56
6.3.2 Data accuracy after replay .........................................56
6.3.3 Recovery time .........................................................57
6.3.4 Storage requirements ...............................................57
6.3.5 Scalability ............................................................60
6.3.6 Implementation complexity .......................................62
6.3.7 Integration complexity .............................................63
6.3.8 Maintainability ........................................................63
7 Conclusions ........................................................................................................65
  7.1 Evaluating goals and outcome ........................................................................65
  7.2 Results and experiences ..................................................................................66
    7.2.1 Evaluating results .................................................................................66
    7.2.2 Improving the quality of the performance test .................................68
    7.2.3 Unexpected problems ...........................................................................69
  7.3 Limitations with the proof-of-concept implementation ............................70
  7.4 Suitability for Bolagsverket ..........................................................................70
    7.4.1 Recommendation .................................................................................72
  7.5 Societal aspects .............................................................................................73
  7.6 Ethical aspects ...............................................................................................73
  7.7 Future work ....................................................................................................74
    7.7.1 Improving performance .........................................................................74
    7.7.2 Other nondeterministic events ...............................................................75
    7.7.3 Undo and Redo ......................................................................................76

References.............................................................................................................77

Appendix A: Relational schema for the databases .............................................83
Appendix B: Measurement data ..........................................................................84
Terminology

Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ATM</td>
<td>Automated Teller Machine</td>
</tr>
<tr>
<td>CIC protocol</td>
<td>Communication-Induced Checkpointing protocol</td>
</tr>
<tr>
<td>COBOL</td>
<td>Common Business-Oriented Language</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>DBA</td>
<td>Database Administrator</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
</tr>
<tr>
<td>e-Government</td>
<td>Electronic Government</td>
</tr>
<tr>
<td>E-ID</td>
<td>Electronic Identification</td>
</tr>
<tr>
<td>E-service</td>
<td>Electronic Service</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HP</td>
<td>Hewlett-Packard</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated development environment</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>Java EE</td>
<td>Java Platform, Enterprise Edition</td>
</tr>
<tr>
<td>Java SE</td>
<td>Java Platform, Standard Edition</td>
</tr>
<tr>
<td>JBoss EAP</td>
<td>JBoss Enterprise Application Platform</td>
</tr>
</tbody>
</table>
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket  
Max Sjölinder  

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK</td>
<td>Java Development Kit</td>
</tr>
<tr>
<td>JMS</td>
<td>Java Message Service</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JPA</td>
<td>Java Persistence API</td>
</tr>
<tr>
<td>JSF</td>
<td>JavaServer Faces</td>
</tr>
<tr>
<td>MOM</td>
<td>Messaging-Oriented Middleware</td>
</tr>
<tr>
<td>MQ</td>
<td>Message Queue</td>
</tr>
<tr>
<td>MTTF</td>
<td>Mean Time to Failure</td>
</tr>
<tr>
<td>MTTR</td>
<td>Mean Time to Repair</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PWD assumption</td>
<td>Piecewise Deterministic assumption</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>RAID</td>
<td>Redundant Array of Independent Disks</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFB</td>
<td>Registrera Företag hos Bolagsverket</td>
</tr>
<tr>
<td></td>
<td>English translation: Register companies at Bolagsverket</td>
</tr>
<tr>
<td>ROC</td>
<td>Recovery Oriented Computing</td>
</tr>
<tr>
<td>SNR</td>
<td>Svenskt Näringslivsregister</td>
</tr>
<tr>
<td></td>
<td>English translation: Swedish Industry Directory</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>vCPU</td>
<td>Virtual Central Processing Unit</td>
</tr>
</tbody>
</table>
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

<table>
<thead>
<tr>
<th>Terminology</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
</tr>
<tr>
<td>XHTML</td>
</tr>
<tr>
<td>XML</td>
</tr>
<tr>
<td>3R</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VM</th>
<th>Virtual Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>XHTML</td>
<td>Extensible HyperText Markup Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>3R</td>
<td>Rewind, Repair and Replay</td>
</tr>
</tbody>
</table>
1 Introduction

The thesis will evaluate two alternative approaches that could be used by the Swedish Companies Registration Office to prevent the loss of customers’ data due to a certain set of unforeseen events. The investigated approaches will, in certain situations, make it possible to recover the correct data from a database containing erroneous or inaccurate data.

The two alternative approaches discussed and evaluated in this thesis, take some precautionary measures during an IT system’s normal operation in order to enable the correct data to be recovered if a problem occurs. The solutions also take advantage of the existing recovery mechanisms of the databases. Proof-of-concept implementations of the two approaches will be developed in order to evaluate the result and determine whether they might prove to be good candidates for the Swedish Companies Registration Office.

1.1 Background and problem motivation

Historically, taking measures to prevent data loss in the case of disasters or other unforeseen events has been vital for organizations. During recent years, a number of cases, where companies have unsuccessfully managed to do so, has attracted considerable attention.

In the beginning of 2008 a technical fault in one of TeliaSonera’s\(^1\) mail servers affected 300,000 email accounts and resulted in customers losing several weeks of incoming emails [1]. In November 2011, a hardware failure in Tieto’s\(^2\) data storage equipment affected approximately fifty of the company’s customers, some of which provided critical public services. The affected organizations experienced varying degrees of business operation disruptions and data loss. [2]

---

\(^1\) TeliaSonera is a company that provides telecommunication and network access services.

\(^2\) Tieto is a company that provides IT related services to customers within the private and public sector.
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

Introduction

The thesis investigates two alternative approaches that, in certain situations, might solve an issue faced by the Swedish Companies Registration Office. The problem could possibly result in data losses of similar magnitudes to those mentioned previously. It should be strongly emphasized that the thesis considers the worst case scenarios and the likelihood that the problem will arise at the Swedish Companies Registration Office is expected to be highly unlikely. Nevertheless, if this does occur the consequences could be very severe, possibly resulting in legal, economic and reputational implications.

The Swedish Companies Registration Office is a Swedish authority that provides its customers with business related electronic services (E-services), including the capability to register new companies, register corporate mortgages and file annual reports [3]. The E-services consists of a large number of cooperating distributed systems that communicate by passing messages and which stores information in databases. The remainder of this thesis will, for simplicity, utilize the alternative name “Bolagsverket”, when referring to the Swedish Companies Registration Office.

The mentioned problem arises because of unforeseen or exceptional events that could cause the data within Bolagsverket’s databases to be disarranged, be erroneous or be inaccurate. This would not be a problem if the unforeseen event was discovered immediately and the changes made to the database could be rolled-back directly. However, if this was not discovered until sometime later, the situation would most certainly be completely different. It would be very likely that new operations would have had time to be executed on the database before the error was detected. This basically means that new information has been inserted, updated or deleted from the database after the unforeseen event had caused the erroneous changes to the database. There is no obvious means of resolving this problem without losing data. The database could, of course, be rolled-back and restored to the point in time just before the error occurred. This would, however, result in losing all perfectly valid changes and new information inserted in the database after the error had occurred. On the contrary, if no action is taken, the database will remain in a state in which it contains disarranged, erroneous or inaccurate information. This would, in turn, imply that
Bolagsverket bases its decisions- and provides its customers with untrustworthy information, which is unacceptable.

Human-made faults are expected to be the leading cause of such a problem. Bolagsverket has identified a number of potential threats, including a database administrator (DBA) making changes to a database which results in unexpected outcomes and batch-programs (a series of sequentially executed programs) that are run in the wrong order causing data to be disarranged.

Bolagsverket actually experienced the latter fault, firsthand. Approximately ten years ago, a system maintainer accidently ran batch-programs in the wrong order, which resulted in one of Bolagsverket’s databases ending up containing erroneous and inaccurate data. Fortunately, the mistake was discovered almost immediately and the surrounding systems could be stopped directly. However, even though Bolagsverket’s staff eventually managed to manually recover the data, it took 20-30 employees, each working approximately 15-30 hours to correct the fault. Imagine the consequences if the mistake had not been discovered until after one hour? One day? Or even a week?

A solution to this problem would provide Bolagsverket with a more robust and recoverable distributed system thus, reducing the risk of losing information. This would, in turn, result in more reliable E-services for Swedish businesses and thereby also benefit society. A solution would most certainly also be valuable for current and future organizations, whose distributed systems are becoming increasingly more complex, while the tolerance for lost- and inaccurate information is decreasing.

Recovery in distributed systems is not in any way a new subject and extensive research has been conducted in the area. However, while the individual techniques used within the thesis have been subjected to a large amount of research, no research has been found that combines the techniques in a manner which is similar to this thesis.

In addition, very little research has been found that actually takes real business- and organization requirements as the starting point for the study. Much of current research appears (somewhat coarsely) to firstly develop a solution before attempting to apply general business needs to
it. This thesis does the opposite: it takes real business needs and develops solutions with respect to these. This is of particular relevance to organizations with systems that already exist but which, initially, were not built to be recoverable and which may be too expensive to rebuild. Thus, organizations such as Bolagsverket must build solutions around existing systems and constraints.

1.2 **Overall aim**

The overall aim of the thesis is to investigate the suitability for utilizing the two alternative approaches referred to as automatic-replay and semi-automatic replay to tackle a problem faced by Bolagsverket. The problem consists of recovering the correct data, in a database that contains inaccurate or erroneous data due to some past fault. The two alternative approaches discussed and evaluated in this thesis take some precautionary measures during a distributed system’s normal operation in order to enable that the correct data can be recovered if a problem arises.

In order to evaluate the alternative approaches in a realistic manner, a simplified and general model of Bolagsverket’s systems is implemented. The model will be used to implement the two approaches in a proof-of-concept manner. The proof-of-concept implementations will be evaluated by conducting performance tests and by means of a theoretical evaluation. An important requirement is that the model should not only be general to Bolagsverket’s systems, but to all systems with similar characteristics to those at Bolagsverket. Thus, the results produced by the thesis should be applicable to both Bolagsverket and other similar systems.

The outcome of the study hopefully indicates that at least one of the approaches is suitable for Bolagsverket. It is also hoped that the results of this thesis are valuable not only to Bolagsverket, but to any organization that experience similar problems.

1.3 **Scope**

The main focus of this thesis is on fault tolerance, using a recovery based approach. This basically means that the thesis will focus on how to avoid failures by recovering from errors. The actual error detection, hence the detection of disarranged, erroneous or inaccurate data within a database is not considered in this thesis. Further, it is not within the
scope of the thesis to repair the mechanism that caused the error. The project will rather aim to recover from the error.

The time-frame within which the recovery procedure will be able to recover the data in the database will be limited to one day back in time. It is, however, assumed that it will be a relatively straightforward task to extend the time-frame in the future.

It is not within the scope of the thesis to consider recovery in all parts of Bolagsverket’s systems. Therefore the project will be limited to analyzing one of Bolagsverket’s E-services and transferring the main characteristics on to a simplified and general model. It is, however, expected that similar characteristics are found in other parts of Bolagsverket’s systems, thus making the results also applicable to the other systems.

The thesis will not consider any security related aspects and will also not consider the situation where the recovery system is susceptible to the same kind of faults it is designed to tackle. Thus, considering how to “recover the recovery system” is not within the scope of the thesis.

When an error has occurred in one system and the database contains disarranged, erroneous or inaccurate data, it is very likely that erroneous data will be propagated to other distributed systems and databases, possibly triggering cascading effects. This is an inevitable and possible consequence when systems utilize the information of other systems during its execution. The thesis will only focus on recovering a single system’s data, without considering any error propagation or cascading effects.

The two approaches investigated in the thesis rely heavily on the assumption that Bolagsverket’s distributed systems are deterministic with respect to the messages passed between the systems. This means that a system always produces the same output, given that the input (the received message) is the same and the state of the system is the same. The thesis does not have access to Bolagsverket’s system and cannot actually verify that the systems are deterministic. It relies entirely on the information and assurance provided by Bolagsverket regarding this property. The only nondeterministic events in Bolagsverket’s distributed system are assumed to be message deliveries. It should be noted that there is always the possibility that nondeterministic events such as
input/output (I/O) interrupts and clock events can affect the execution of a system. These kinds of exceptional events are not considered in the thesis.

Finally, the thesis is limited to utilizing the same technologies as used by Bolagsverket.

### 1.4 Concrete and verifiable goals

The thesis will assess the suitability for utilizing the two alternative approaches referred to as automatic-replay and semi-automatic replay, in order to enable Bolagsverket to recover the correct data from a database that belongs to a stateless distributed system and that contains erroneous- or inaccurate information due to past faults. This is conducted by developing proof-of-concept implementations of the two alternative approaches on a simplified and general E-service model of Bolagsverket’s IT systems. The proof-of-concept implementation will be evaluated both theoretically and by means of performance tests.

The concrete goals of the thesis are divided into the three main categories: E-service model, proof-of-concept implementation and evaluation. Each category is presented in further detail below.

A simplified and general E-service model that mimics one of Bolagsverket’s E-services will be implemented in order to be able to test the proof-of-concept implementation in a realistic environment. An analysis of Bolagsverket’s IT systems will be conducted to define the exact characteristics, function and setup of the model. However, in addition to resembling Bolagsverket’s system, it is important that the model remains general in order for the thesis to produce general results. More specifically, the concrete low-level goals regarding the development of the E-service model are:

- Requirements for the E-service model, derived from the analysis.
- An implementation of a simplified and general E-service model.
Automatic- and semi-automatic replay will be developed as proof-of-concept implementations using the E-service model. The concrete low-level goals for the proof-of-concept implementation are:

- A logging system that is utilized for logging messages to persistent storage. The logging system should be integrated with the E-service model so that all nondeterministic events in the model are logged.

- The functionality to restore the database used in the E-service model to a previous point in time.

- A replay system, implementing the following two approaches to data recovery:
  
  o **Automatic replay**: replay logged messages in the same manner as they were sent during the original execution. No human intervention should be required during the replay procedure.
  
  o **Semi-automatic replay**: utilize the logged messages to enable the official’s at Bolagsverket to manually redo lost work. During normal operation, officials conduct work by handling errands. The aim of the replay procedure is to make it possible for officials to redo work by re-handling errands, as in the case during the original handling procedure. The result of the re-handling process should be exactly the same as the result of the original handling process, given that the officials submit the information suggested by the recovery system.

The proof-of-concept implementation will be evaluated by conducting performance tests/calculations and a theoretical evaluation. The concrete low-level goals are to conduct the evaluation with respect to the following properties:

- Theoretically evaluated properties:
  
  o Scalability
  o Implementation complexity
  o Integration complexity
  o Maintainability
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket
Max Sjölinder

1 Introduction

• Measured/calculated properties:
  ○ Data accuracy after replay
  ○ Logging overhead
  ○ Recovery time
  ○ Storage requirements

1.5 Outline
Chapter 2 describes the theory concerning dependable systems, particularly with regards to rollback-recovery in distributed systems. The chapter also describes technologies relevant to the thesis, as well as related research and work. Chapter 3 analyses the parts of Bolagsverket’s IT systems that are used to handle the registration of new companies. Chapter 4 describes the methodology used for the performance- and theoretical evaluation. The chapter also presents the software and hardware resources utilized in the thesis. Chapter 5 describes the implementation of the E-service model and the proof-of-concept solutions. Chapter 6 analyses whether the goals set for the implemented software have been met. The chapter also presents the outcome of the performance test and the theoretical evaluation. Chapter 8 concludes by discussing the results and the outcome of the thesis. The chapter also discusses the suitability for Bolagsverket to utilize the two investigated approaches. Finally, societal- and ethical aspects, as well as future work are presented.

1.6 Contributions
All work in the thesis has been conducted by the author. Bolagsverket has, during the course of the thesis, however, contributed with information and statistics regarding their systems. This information have then been interpreted and presented by the author.
2 Theory

The chapter presents information concerning the theory, techniques and technologies related to the thesis. The chapter begins with a brief introduction to distributed systems. The chapter also covers dependable systems and fault tolerant computing, especially with regards to rollback-recovery in distributed systems. Further, the chapter briefly presents information regarding the Java Platform, Enterprise Edition (Java EE) and restoring/recovering a database to a previous point in time. Finally, related work and research is covered.

Much of the information provided in the chapter is considered as prerequisite knowledge for the subsequent chapters.

2.1 Distributed systems

A distributed system can be defined as a set of autonomous computers, connected to a network that communicate and coordinates its operation by passing messages. The computers may be situated at the same location or spread out globally. Systems add value to the user (people or programs) by providing meaningful services, often in the form of shared resources. Programs executing on distributed computers are often referred to as processes. [4] Processes communicate and organize themselves exclusively using messages. The state of a system is said to be consistent if, for every process that states that it has received a message, there also exists a process which states that it has sent that same message. [5]

Some significant aspects that should be considered when dealing with distributed systems are: concurrency, lack of a global clock and independent failures [4]. Processes across the system will run concurrently and thus, there is a need to coordinate processes that share resources. Further, since processes can be executed on different machines, each having its own physical clock, the notion of a specific point in time is problematic due to deviations between clocks. It is not possible to perfectly synchronize all clocks in distributed systems [6], it is therefore often valuable to utilize the happened-before relation proposed by Leslie Lamport [7], when considering the partial order of events be-
between distributed processes. Hardware and software components in individual systems can fail in numerous ways, and it is crucial to consider how failures will affect the distributed system.

2.2 Dependable systems

Avizienis et al. [8] defines dependability as the “ability of a system to avoid service failures that are more frequent or more severe than is acceptable”. Dependability consists of the characteristics availability, reliability, safety, integrity and maintainability. A service failure (often called merely “failure”) occurs whenever a service is unable to deliver the correct function to the user. An error is a system’s inability to perform the correct function, but it should be noted, however, that a failure only occurs when a user (or equivalently, the system’s external state) experiences the error. The source of an error is called a fault, and can either be internal or external to the system. There is a wide range of different kinds of faults that can affect the system, including human-made faults which are derived from human actions. [8]

Human-made faults are one of the major causes of failures in computer systems. A survey conducted on the U.S. Public Switched Telephone Network by Patterson et al. [9] showed that human operators were to blame for more than half of all telephone outages. Another report suggests that human error and software failures account for 80 % of all web application failures [10].

There are four different categories of approaches for achieving more dependable systems: fault prevention, fault tolerance, fault removal and fault forecasting. Only fault tolerance is covered here and the interested reader is referred to the paper by Avizienis et al. [8] for more information regarding the other three categories. Fault tolerance deals with failure avoidance, using error detection and recovery. Error detection is basically concerned with the detection of errors in a computer system, while recovery is concerned with transforming a system containing errors into a valid and correct state. Recovery can further be classified as fault handling, which aims to prevent prior faults from being re-activated, and error handling, which aims to remove an erroneous state from the system. [8] This thesis is mainly interested in recovery from errors and failures resulting from human-made faults in message passing distributed systems, more specifically, those resulting in data loss.
2.2.1 Error handling approaches
Rollback- and rollforward recovery are error handling techniques that take two distinct approaches for achieving error recovery [8]. Rollforward recovery aims to transfer a system from an erroneous state, directly to a new valid state and proceed with the execution from this point. A disadvantage associated with the approach is that, the errors that might occur have to be anticipated in advance, in order to make the transition to a new valid state. In contrast, rollback recovery aims to transfer the system from an erroneous state back to a previously valid state from which the system can proceed with its execution. Rollback recovery is a popular and frequently adopted rollback recovery technique. This is mainly because the technique is very general and can be adopted in a variety of situations.[6] A disadvantage associated with the rollback-recovery technique is its inability to recover from design faults. Utilizing this approach, when attempting to recover from a failure or error due to a design fault, will result in the error or failure only repeating itself. [11]

The thesis has its focus on rollback-recovery in distributed systems and the next subchapter is devoted to the subject.

2.3 Rollback-recovery in distributed systems
Rollback recovery techniques can be classified as being either checkpoint-based or log-based. Checkpoint-based recovery means that checkpoints are taken and saved at regular intervals during the normal execution of a process. A checkpoint is a complete “snapshot” of the state of the process at a given time. During recovery, the distributed system can be rolled back to the most recent consistent set of checkpoints (called recovery line) and the execution can be resumed from this point. Log-based recovery regularly takes checkpoints, as well as logging nondeterministic events that are executed by the process. During recovery, the process can be rolled back to the most recent checkpoint and the logged nondeterministic events can be re-executed. Log-based recovery enables the recovery of the state of the process just prior to the occurrence of the error or fault. In contrast, checkpoint-based recovery does not guarantee that the process is recovered to the state just prior to the occurrence of the error or fault. When discussing rollback-recovery, it is expected that the processes have access to stable storage and that the storage can withstand all kinds of failures and errors that a process is required to
recover from. The storage is used to save the information that is required for recovery. It should be noted that a system’s interactions with the outside world (also called the systems external state) generally cannot be relied upon to be rolled back. An automated teller machine (ATM) could, for example, not be rolled back when a client’s money already have been ejected. [12]

The following subchapters will cover checkpoint-based and log-based rollback recovery in further detail.

2.3.1 Checkpoint-based rollback-recovery
As stated previously, checkpoint-based rollback-recovery takes regular checkpoints during the normal execution of a process in order for the system to rollback and proceed with the execution from the recovery line, in the case of failure. Figure 1 illustrates a situation where process P2 has failed and the system will proceed from the most recent set of consistent checkpoints (the recovery line). It should be noted from the figure that some of the checkpoints are essentially useless since they are unable to form a consistent recovery line.

![Figure 1: Illustrating the recovery line. [6]](image)

Checkpoint-based rollback-recovery can suffer from the domino effect, which can result in losing large amounts of work. This is essentially the situation in which several processes have to be rolled back due to the failure of one process, which in turn can trigger other processes to be rolled back and so on. This occurs when the global state of the processes is inconsistent after the initial rollback, and other processes have to be rolled back in order to obtain a consistent global state. For example, if a sender of a message is rolled back to the time before a message is sent,
the receiver of the message must also roll back to the time before the message was received, since it cannot receive a message that no one has sent. The domino effect is caused by the individual processes taking checkpoints in an uncoordinated manner. [12] Figure 2 illustrates this in a situation with two processes, where process P2 has failed and the domino effect will cause both processes to roll back to the initial state since that is the most recent consistent recovery line [6].

![Figure 2: Illustrating the domino effect. [6]](image)

The three main approaches for checkpoint-based recovery are briefly presented below.

**Uncoordinated checkpoint-based rollback-recovery**

In uncoordinated checkpoint-based recovery, the processes in the distributed system may take checkpoints whenever they please. Even though this may appear to be advantageous due to its simplicity, there are several drawbacks. The approach suffers from the domino effect, thus work can be lost. Further, there is the possibility that a process takes useless checkpoints that only incur unnecessary overhead for the process, see Figure 1. [12]

**Coordinated checkpoint-based rollback-recovery**

In coordinated checkpoint-based recovery, the processes coordinate their checkpoints in such way that they always form a consistent set of checkpoints, and thus a recovery line. The overhead involved in coordinating the checkpoint is a drawback associated with this approach. However, the approach does not experience the domino effect and recovery is simple, since the processes merely rollback to their most recent checkpoint. [12]
Communication-induced checkpoint-based rollback-recovery

In communication-induced checkpoint-based recovery, a communication-induced checkpointing (CIC) protocol is used to avoid both the domino effect and the requirement for global coordination. Processes may take uncoordinated checkpoints referred to as local checkpoints. The CIC protocol may also force the processes to take forced checkpoints in order to avoid the domino effect. [12]

2.3.2 Log-based rollback-recovery

Log-based rollback-recovery takes advantage of the processes deterministic behavior in order to perform recovery. The execution of a process can be seen as a number of separate intervals, which are executed sequentially. The intervals either start with a nondeterministic event or with the initial state of the process, and the remainder of the interval consists of the deterministic states of the process. Thus, by saving the ordered set of all nondeterministic events it is possible to reproduce the execution of the whole process. In distributed systems, received messages are often seen as nondeterministic events. [13] Log-based recovery is based on the piecewise deterministic (PWD) assumption, which states that all nondeterministic events can be detected and logged in such way that the events could be recreated and replayed during recovery. When the nondeterministic events are encoded and logged they are referred to as determinants. As stated previously, log-based rollback-recovery makes use of both checkpoints and determinants in the recovery procedure. The checkpoints are used to limit the amount of logged events that have to be replayed during recovery. Thus, during recovery, the process can be rolled back to the most recent checkpoint before replaying all determinants from this point.

In message passing systems where message receipts are nondeterministic events, the logging can either be receiver-based or sender-based. The main difference lies in whether the determinants are logged by the receiver or the sender process. The sender-based logging approach is often considered to have a better performance since it is possible to send the message prior to the message being logged. [12]

Log-based rollback-recovery experiences a performance overhead during normal, failure-free operation due to the determinants being logged. The amount of overhead does, however, depend on the ap-
proach being used. In contrast to checkpoint-based rollback-recovery, log-based recovery does not have the problem of domino effects. However, depending on the approach used for logging determinants, it can suffer from orphan processes. [12] Orphan processes are processes in which the state is inconsistent with the state of a process that has recovered from failure. Orphan processes are required to be rolled back in order to transfer the system into a consistent state. [13]

The three main approaches for logging nondeterministic events, also referred to as message logging, are presented below.

**Pessimistic logging**

In pessimistic logging, a process synchronously logs every determinant before it can be used in the execution of the process. The logging approach is called pessimistic since it assumes that failures can occur at any time. The approach does not suffer from orphan processes and it enables a simple recovery procedure. During recovery, the failed process is the only process that is required to be rolled back and re-executed. This is important since the failed process can recover all by itself, without the necessity to coordinate the recovery procedure with other processes. Further, the failed process only requires to be rolled back to the most recent checkpoint. This basically means that only one checkpoint has to be stored for each process. However, the performance overhead during failure-free operation can be substantial since the execution of the process is blocked until the determinants have been logged. Nevertheless, if an application can tolerate the extra overhead, pessimistic logging is an attractive alternative due to its simplicity. [12]

**Optimistic logging**

In optimistic logging, a process asynchronously logs the determinants to stable storage. This is performed by firstly saving determinants in volatile storage, before periodically saving these to stable storage. The logging approach is called optimistic since it assumes that the determinants will be logged to stable storage before a failure has occurred. Optimistic logging has a better failure-free performance as compared to pessimistic logging. This is due to the reduced failure-free overhead resulting from avoiding blocking the execution of the processes when writing to stable storage. However, recovery with optimistic logging is more complex. If
a process fails it will lose all determinants saved in volatile storage and thus the process will fail to reproduce these nondeterministic events. As a consequence, orphan processes will be created, requiring other processes to be rolled back to a consistent state. [12]

**Causal logging**

Causal logging combines beneficial characteristics from both pessimistic- and optimistic logging. The approach does not suffer from orphan processes and has a failure-free performance similar to that of optimistic logging. The drawback associated with causal logging is that it has a much more complex recovery procedure than the other logging approaches. [12][14]

2.3.3 Programming transparency

When implementing rollback-recovery facilities, it is important to consider the extent to which reliance should be placed on the application programmer to ensure that checkpointing and logging is performed correctly. Leaving the huge responsibility of taking correct checkpoints and determinants to the application programmer is risky since it is susceptible to human-made faults. Depending on the approach chosen for achieving rollback-recovery, the application developer will experience different degrees of programming transparency.

Application-level checkpointing and logging could be used to enable a programmer to insert the necessary checkpoint/logging code directly into the program’s source code. It could also be conducted at a user-level by linking libraries to the program that handles the issue. Further, rollback recovery can be implemented in system middleware’s. Alternatively, checkpointing and logging could also be performed on the kernel- or hardware-level, thus being totally transparent to the application programmer. [12][14][15]

2.4 Java Enterprise Edition

Java Platform, Enterprise Edition (Java EE) is built on top of the Java Platform, Standard Edition (Java SE). It enables developers to build large scale, multi-tiered, scalable, secure and reliable applications.
Java EE is comprised of a number of technologies, frameworks and Application Programming Interfaces (APIs), some of which are relevant to the thesis and which are briefly presented in the following section.[16] Note, however, that some of the technologies described subsequently are part of the Java SE platform.

2.4.1 JavaServer Faces

JavaServer Faces (JSF) is a web tier technology that can be used to ease the development of server-side user interfaces. Web components such as buttons and input fields are added to web pages and are connected to server-side objects using tag libraries. JSF is also used to handle the state of components, validate and convert component data, handle events and page navigation.[16][17]

Components in a JSF web page are often associated with Managed Beans that validate the component’s data, handle events and determine page navigation. Component instances and component values are usually bound to properties within the Managed Bean. [18]

2.4.2 Java Message Service

Java Message Service (JMS) is a peer-to-peer (P2P) messaging API that enables Java applications to communicate with other messaging-oriented middleware (MOM). JMS enables a loosely coupled communication among applications, which basically means that the sender and receiver have to know practically nothing about each other in order to communicate. Further, the receiver does not even have to be up and running when the message is sent, and vice versa. JMS also facilitates once and only once message delivery and enables an asynchronous communication model. [19] The message objects sent between applications, containing the information desired to be transferred are called JMS messages [20].

2.4.3 Enterprise beans

Enterprise beans are server-side components based on the Enterprise JavaBeans (EJB) technology that enables developers to build distributed, secure and transactional Java applications. Enterprise beans are used to implement the core functionality of an application, often called business logic. Enterprise beans can be of two types: session beans or message-driven beans.[21] [22]
Session beans can either be stateful or stateless, which basically means that between subsequent calls to the bean, the bean either stores or does not store, the state about the individual calling clients. The methods in the session bean that should be accessible by clients (called business methods) are defined in an interface called the business interface.[23] [24] Message-driven beans are invoked through messages and enable applications to communicate asynchronously. The message-driven bean typically consumes JMS messages, even though it also supports other kinds of messages. The messages can be sent to the message-driven bean directly by a client or alternatively, the bean could fetch the message from e.g. a JMS queue. [25]

2.4.4 Java Naming and Directory Interface
The Java Naming and Directory Interface (JNDI) enables resources such as databases and enterprise beans, to be located by looking them up using names. JNDI makes it, for instance, possible for a Java client to look up the JNDI name of an enterprise bean and invoke some of its business methods. [26]

2.4.5 Java Persistence API
Java Persistence API (JPA) is a specification that enables object-relational mapping for relational data. JPA provides developers with a convenient way of persisting, updating and deleting objects, known as entities, from storage. A JPA entity is a Java class that often represents a table in a relational database. A single row in the table is represented by an instance of the entity class. [27]

2.4.6 RESTful web services with JAX-RS
JAX-RS is an API that enables the development of web services that conforms to the representational state transfer (REST) architecture. The API can also be used for accessing existing RESTful web services as a client. [28]

2.5 Database recovery and restoration
The chapter is only concerned with recovering and restoring a database to a previous point in time. There are several ways of achieving this, some of which are vendor specific. It should be noted that the approaches presented here are not in any way exhaustive.
One way of recovering a previous version of a database is to restore a backup. Even though the approach is intended to recover the database in the case of catastrophic failures, it could also be used for other purposes. Another way is to utilize the database system log in order to undo transactions. [29] If Oracle databases are used, a technology called “Flashback” can be utilized. The Flashback technology enables the viewing of past states of data, as well as rewinding and winding data backwards and forwards in time. Using Flashback, a single or multiple tables could be recovered to a previous point in time. [30]

2.6 Related work

There has been extensive work and research conducted in the area of fault tolerance via rollback-recovery and replay mechanisms. The chapter will present some of this work, but also some related research fields that bear resemblance- or have relevance to the thesis.

2.6.1 Recovery oriented computing

Patterson et al. [9] suggest that the computing industry should change its attitude and mindset toward hardware faults, software bugs and operator errors. The authors propose recovery oriented computing (ROC) which approaches faults, errors and failures as facts that must be coped with, instead of approaching these as problems that must be solved. Patterson et al. suggest that systems should be able to fail- and recover rapidly. The authors also suggest that focus should be shifted from the mean time to failure (MTTF) metric, to the mean time to repair (MTTR) metric. Further, Patterson et al. [9] observe opportunities in the virtual world of computers that are very different in relation to the physical world. The authors write:

“Civil engineers must design walls to survive a large earthquake, but in a virtual world it may be just as effective to let it fall and then replace it milliseconds later, or to have several walls standing by in case one fails.”

2.6.2 Logging and checkpointing tools

Much research and work has been conducted in the area of logging and checkpointing, resulting in several available tools. Egwutuoha et al. [11] Have collated a list of different checkpoint/ restart facilities. Further, Maloney, Goscinski [31] have reviewed a number of checkpointing
facilities, mainly with respect to cluster systems. A few of the tools will be covered, briefly, below.

DMTCP is a software package that can be used for transparent user-level checkpointing of distributed applications. DMTCP also has support for checkpointing cluster computations. [32]

DejaVu is a transparent user-level tool for fault tolerance in parallel and distributed applications. It automatically takes checkpoints and can recover from any combination of system failures. [33]

CRAK is a transparent kernel-level checkpoint/restart facility for Linux. It neither requires any modifications to the application nor to the kernel. The main use of the tool is for application migration. [34]

2.6.3 Undo and Redo
Research has been conducted exploring the possibility of undoing and redoing past events in systems. This basically makes it possible to “travel in time” and fix past problems. This kind of mechanism is particularly attractive for humans, since it supports trial-and-error reasoning at a system level. [35]

A model called Rewind, Repair and Replay (3R) has been proposed that enables errors from the past to be corrected using a three step approach. During the rewind step, all of the system’s state, down to the operating-system level, is restored to an earlier point in time. During the repair step, any changes to the system could be made. Finally, the replay step re-executes all user interactions with the system. The 3R model tracks the user interaction by logging the user’s intent with so called “verbs”. [36] Joyce is another tool with similar functionality to that of the 3R model. It enables a system’s history to be safely navigated, edited and experimented with, by a user.[37]

2.6.4 Related research fields
There are several other research fields that have similarities to the thesis. It should be noted however that the thesis has only briefly investigated these. The chapter should, instead, be seen as an overview and a starting point for further research.
Research in the area of distributed debugging has close resemblance to checkpointing and logging in rollback-recovery. Distributed debugging and rollback-recovery techniques consider nondeterministic events in a similar fashion. By recording a program’s original execution it is possible to replay and debug a program’s execution at a later time. Two examples of library-based tools for distributed debugging are R2 and Jockey. [38][39]

Research in the field of dirty data and data cleansing have similarities to the problem domain of the thesis. Dirty data concerns data sources that contain missing data, wrong data, and different representations of the same data [40]. Data cleansing concerns identifying and eliminating impurities and irregularities (such as dirty data) in data [41].

Research has also been conducted that utilizes rollback-recovery techniques to bypass software faults. This is performed by a technique called progressive retry, where logged messages are reordered and replayed, and the scope of the rollback is gradually increased. [42]
3 Analysis of Bolagsverket’s IT systems

The parts of Bolagsverket’s IT systems that handle registrations of new companies is analysed in the chapter. The analysis mainly focuses on the basic function and characteristics of the systems. The chapter firstly presents an overview of the system, before giving a somewhat more detailed description.

3.1 Overview

Bolagsverket provides its customers with business related E-services, including the capability to register new companies, register corporate mortgages and file annual reports. The E-services consist of a large number of distributed systems providing different functionalities. At the core of these systems is a management system called “Unireg”, which consists of approximately 2,000 programs written in the Common Business-Oriented Language (COBOL). Unireg provides Bolagsverket with much of its basic operational- and IT support systems, including an official’s ability to process errands and customer’s ability to indirectly perform business related tasks.

A simplified illustration of the system’s basic operation and its interaction with customers can be seen in Figure 3. Customers can provide information to Bolagsverket either by sending in physical documents (as customer 1), or by signing in using electronic identification (E-ID) and providing the information digitally (as customer 2). The sent in physical documents are scanned and sent to Unireg, where they are processed either automatically or manually by officials. The scanned documents are also sent to an E-archive system called “Billy”. Customers with electronic ID can, via a website called “verksamt.se” [43], interact with an E-service called RFB. RFB consists of several systems and provides the customers with different functionalities, including the functionality to register new companies. RFB communicates with Unireg and stores information in the E-archive system, “Billy”.

22
Bolagsverket also provides customers (customer 3) with the ability to buy information about companies. This is conducted via an E-service called SNR (in Swedish: Svenskt Näringslivsregister) [44]. There are also customers who subscribe to the latest information from Bolagsverket. One such customer is UC [44], which sells business- and credit information. The latest information is sent to subscribing customers such as UC, continuously using the File Transfer Protocol (FTP).

Fault tolerance for the databases is provided using several techniques. Data redundancy and data availability is provided by utilizing redundant arrays of independent disks (RAID) solutions, as well as mirroring the databases to a geographically separate location. Offline backups are also taken at regular intervals. Scalability is achieved by load balancing between several servers running the same software.

3.1.1 Migration from a mainframe architecture

Bolagsverket’s systems were initially developed for a mainframe architecture, consisting of a mainframe computer and thin clients acting as terminals. The terminals were used by officials to manage errands. In fact, Unireg was run on a mainframe computer until 2012 when Bolagsverket migrated to an open systems architecture based on an x86 central processing unit (CPU) architecture. The migration resulted in less vendor dependent systems, thus increasing the flexibility in the types of
technologies that could be utilized. The migration was conducted by Hewlett-Packard (HP) and the aim was to have as small an impact on existing systems as possible, since it was not feasible to rewrite all existing software for the new architecture. This effectively means that Unireg currently runs on a system that emulates a mainframe computer and the terminals are run on emulated thin clients.

### 3.2 Company registration via RFB

The chapter analyzes how customers apply to register companies at Bolagsverket via the E-service RFB and the user interface (UI) “verksamt.se”. RFB consists of several collaborating systems, which provide the functionality utilized by the customer via “verksamt.se”. These systems communicate asynchronously using messages and message queues.

Figure 4 presents an architectural model of the main parts involved when customers apply to register new companies via “verksamt.se”. When the customers submit an application to RFB, a JMS message containing the application will be sent to the message queue. Unireg will fetch the message from the queue when it has available resources, process the message and store the application in its database. The application will be stored in the database until it is manually handled by an official working at Bolagsverket. An application is often called an errand when it has reached Bolagsverket. The remainder of the thesis will, for that reason, refer to an errand whenever an application has been submitted to the RFB E-service.

It is important to note that the Unireg software is stateless and that it uses a database to store any necessary information. However, Unireg is often referred to as one entity, consisting of both the software and the database. In this case, the system is stateful and all state is stored within the database. The interface toward Unireg is deterministic with respect to the JMS messages. Thus, Unireg will always produce the same output, given that the input is the same and the system (the database) contains the same state. It also satisfies the piecewise deterministic (PWD) assumption covered in the theory chapter.
3.3 **Errand handling**

The chapter analyzes how officials at Bolagsverket handle errands and, more specifically, errands related to the registration of new companies.

As discussed previously, after the migration of 2012 officials handle errands on terminals that are emulating thin clients. Figure 5 presents an architectural model of the main part involved in the errand handling procedure. Officials use console interfaces running on clients which communicate with a web server. The web server maps the HTTP (Hypertext Transfer Protocol) traffic sent from the clients to Unireg specific XML (Extensible Markup Language) messages sent over HTTP. The communication between the officials and Unireg is synchronous.

From a functional point of view, an official can pick available errands stored in the Unireg database and handle these. The official gives each handled errand a unique case number. Further, officials handle errands related to the registration of new companies by verifying the submitted application and by specifying a unique organizational number for the new company.

It is not clearly determined whether the Unireg interface toward the officials is deterministic with respect to the XML messages. Determining this is difficult because of the complexity of the interface and since all code related to the migration is the property of HP. This is also the reason why the thesis examines two alternative approaches. Automatic replay exploits the deterministic behavior of the interface and assumes that the PWD assumption will hold true. In contrast, semi-automatic replay does not exploit the deterministic behavior of the interface. It should be noted that in the case where Unireg is shown to be nondeter-
ministic, it is expected to be possible to modify the system in order to make it deterministic. It is, however, not within the scope of the thesis to consider how this could be accomplished.

![Diagram of system components](image.png)

**Figure 5: Official handling errands.**

### 3.4 Statistics

The chapter presents statistics regarding Bolagsverket’s IT systems and handled errands. Table 1 presents data regarding officials and handled errands during the entire of 2012. Table 2 presents technical data regarding the average size of the messages and the average number of messages that pass through the system each day.

**Table 1: Statistics concerning the whole year of 2012, regarding handled errands at Bolagsverket.**

<table>
<thead>
<tr>
<th>Property</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of handled errands</td>
<td>~ 500 000 errands</td>
</tr>
<tr>
<td>Total number of officials</td>
<td>~ 220 officials</td>
</tr>
<tr>
<td>Average number of handled errands per workday (8 hours)</td>
<td>~ 1923 errands per workday</td>
</tr>
<tr>
<td>Average handling time per errand</td>
<td>~ 0.92 hours per errand</td>
</tr>
</tbody>
</table>
Table 2: Statistics regarding the average size of the JMS- and XML messages.

<table>
<thead>
<tr>
<th>Property</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average JMS message size</td>
<td>~ 100 kilobyte</td>
</tr>
<tr>
<td>Average XML message size</td>
<td>~ 100 kilobyte</td>
</tr>
<tr>
<td>Average number of JMS messages per day</td>
<td>~ 1923 messages per day</td>
</tr>
<tr>
<td>Average number of XML messages per day</td>
<td>~ 1923 x 15 messages per day</td>
</tr>
</tbody>
</table>
4 **Methodology**

The chapter presents the methodology used in the thesis work. This includes the software development process, hardware- and software tools and the evaluation of the proof-of-concept implementation.

4.1 **Software development methodology**

The software was developed using an iterative and incremental software development process. A working, yet very basic version of the software, was firstly developed and tested. The functionality and quality of the software was gradually enhanced during subsequent iterations.

4.2 **Tools**

The chapter presents the tools used in the thesis.

4.2.1 **Hardware**

The chapter presents the hardware used, as well as any virtual machines (VMs). This is achieved by specifying the CPU, Random Access Memory (RAM) and operating systems (OS).

**Development and deployment**

The proof-of-concept implementation as well as the E-service model were developed, deployed and tested on a Hewlett-Packard Compaq dc7900 with the following hardware:

**CPU:** 2 Core Intel CPU E8400 3.00GHz x64

**RAM:** 8.00 GB

**OS:** Microsoft Windows 7 Enterprise x64, Service pack 1

**Database server**

The databases run on a VMware vSphere 5.1 VM consisting of 2 virtual central processing units (vCPUs) and 12 GB RAM. The hardware consists of a HP ProLiant DL380 G7 with the following specifications:
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

4 Methodology

2013-12-27

CPU: 4 Core Intel Xeon X5687 3.60GHz x64

RAM: 98 GB

OS: SUSE Linux Enterprise Server (SLES) 11, service pack 2

Message queue

The IBM WebSphere message queue (MQ) runs on a VMware vSphere 5.1 VM consisting of 1 vCPU and 6 GB RAM. The hardware consists of a HP ProLiant DL385 G7 with the following specification:

CPU: 32 Core AMD Opteron 6282SE 2.6 GHz x64

RAM: 130 GB

OS: Microsoft Windows 2008R2

4.2.2 Development tools and software

The software and tools used during the development are now presented.

Programming languages and platform

The development of the proof-of-concept implementation, as well as the E-service model was conducted by utilizing the Java EE 6 platform with the Java programming language. Table 3 presents the main Java technologies, frameworks and APIs used (some are part of the Java SE platform).

Table 3: The Java technologies, frameworks and APIs used in the thesis.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Version/implementation used</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaServer Faces</td>
<td>JSF 2.0 Mojarra reference implementation, release 2.0.2 [46]</td>
</tr>
<tr>
<td>Java Message Service</td>
<td>JMS version 1.1, as part of Java EE 6</td>
</tr>
<tr>
<td>Enterprise beans</td>
<td>EJB 3.0 specification included in Java EE 5 [47]</td>
</tr>
</tbody>
</table>
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket
Max Sjölinder

4 Methodology
2013-12-27

<table>
<thead>
<tr>
<th>Java Naming and Directory Interface</th>
<th>JNDI version 1.2, as part of the Java SE 6 platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Persistence API</td>
<td>Hibernate, version 3.6.0 Final, was used as the JPA persistence provider [48]</td>
</tr>
<tr>
<td>RESTful web services (JAX-RS)</td>
<td>JAX-RS 2.0 implemented in the RESTEasy framework, version 3.0-rc-1 [49]</td>
</tr>
</tbody>
</table>

The Extensible HyperText Markup Language (XHTML) and Cascading Style Sheets (CSS) were used to create and style web pages. The Structured Query Language (SQL) was used to create and manage the databases.

**Application Server**
The proof-of-concept implementation, as well as the E-service model were deployed and tested on a JBoss application server, part of the JBoss Enterprise Application Platform (JBoss EAP) 4.3.0, cumulative Patch 3.

**Integrated development environments**
Eclipse Helios (service release 2) for Java EE developers was used as the integrated development environment (IDE) for Java programming, with the Java Development Kit (JDK) version 6 update 33.

Oracle SQL developer, version 3.2.20.09 was used as the IDE when working with databases.

**Database**
The thesis utilizes the 64 bit, Oracle Database 11g Enterprise Edition Release 11.2.0.3.0.

**Message queue**
The message queue used during the thesis is the IBM WebSphere MQ, version 7.1.
4 Methodology

Web browsers
Internet Explorer and Google Chrome were the web browsers used throughout the course of the project.

4.2.3 Other software tools

Performance - and functional testing
Apache JMeter was used as a tool to evaluate the performance of the proof-of-concept implementation. The tool was used since it provided an easy means of simulating multiple concurrent users (customers and officials) for testing purposes.

Another tool used for performance testing was Java’s System.nanoTime() method, which was used to measure the execution time for some piece of code. The method was used since it is already part of the Java language and it provided a convenient means of measuring the execution time directly in the code without having to use any third party tools. An important note about the method is that even though it provides nanosecond precision, it does not guarantee nanosecond accuracy [50]. Thus, measurements of this order of magnitude should not be trusted. However, since the maximum time accuracy required in the thesis was of milliseconds magnitude, the method was expected to suffice.

Data analysis
Matlab was used to analyze and plot the data gathered from the performance tests.

Modeling
The open-source tool UMLet was used to create the unified modeling language (UML) class diagrams. High-level illustrations were made using the online drawing and diagram tool Cacco.com.
4.3 System evaluation

The system evaluation aimed to gather objective information such as performance characteristics, about the implemented proof-of-concept logging and replay system. The system was evaluated with regards to the following properties, as stated in chapter 1.4:

- Measured/calculated properties:
  - Logging overhead
  - Data accuracy after replay
  - Recovery time
  - Storage requirements
- Theoretically evaluated properties:
  - Scalability
  - Implementation complexity
  - Integration complexity
  - Maintainability

Whenever it was necessary to simulate a situation in which the database contained inaccurate or erroneous information, this was performed by simply updating and setting an entire column in the database to zero. The following subchapters provide further details regarding the evaluation for each of the previously mentioned properties.

4.3.1 Logging overhead

The overhead with regards to the logging of one message was measured for different message sizes. This was achieved by measuring the time from the point where the logging of one message was initiated until the message was successfully logged. The measurements were conducted on both JMS- and XML messages and the measurements for each data point were repeated 10,000 times. The message size was modified by adding dummy data to the messages. The following message sizes were used: 1 kB, 5 kB, 10 kB, 50 kB, 100 kB, 250 kB, 500 kB, 750 kB, 1,000 kB. The arithmetic mean and standard deviation were calculated and presented. The resulting mean values for the JMS- and XML messages were plotted on a graph. The results were valid for both automatic- and semi-automatic replay since they utilize the same logging mechanism.
4.3.2 Data accuracy after replay

The data accuracy property states whether or not the recovery procedure succeeded in correctly recovering the data in the database. The property essentially results in a yes or no answer: either the data was accurate after replay or it was not. It should be noted that this property was of vital importance when evaluating the implementation. Thus, accurate data after replay was a prerequisite for the system to function correctly.

The data accuracy property was measured by comparing the content in the database (containing correct information) prior to the replay procedure being initialized, with the content in the database after replay. If the content prior to and after replay was identical (they contain exactly the same data) the data was considered to be accurate. The following list presents the steps conducted each time data accuracy was measured:

1. Simulate users utilizing the E-service.

2. Save content in the database to temporary storage

3. Simulate the database containing inaccurate and erroneous data

4. Restore database to previous point and run the replay procedure

5. Compare the database content prior to- and after replay.

The data accuracy measurements were conducted simultaneously with the recovery time measurements, thus for each measured recovery time, the data accuracy was also measured.

4.3.3 Recovery time

The time it takes to recover the data in the database during the replay procedure is referred to as the recovery time or replay time. The recovery time does, however, not include the time it takes to restore the database to a previous point in time, since there are several ways to accomplish this, each of which is expected to have a different performance. Thus, by excluding these measurements, the resulting recovery times are valid independent of the restoring technique. More specifically, in the case of automatic replay, the recovery time was the time interval from when the replay procedure was initiated until the last replayed
messages had been processed by the system. In the case of semi-automatic replay, the recovery time was the time from when the replay procedure was initiated, until the last errand was re-handled by officials, or, alternatively, the last replayed messages had been processed by the system, depending on which of the two was the last to occur.

**Automatic replay**

The recovery time for automatic replay was measured by replaying different amounts of messages, corresponding to data recovery from different points in time. The following numbers of messages were replayed: 100, 1 000, 10 000, 30 000. The messages consisted of equal amounts of JMS- and XML messages. The measurements were also conducted using different message sizes in order to obtain general results. The sizes of the JMS- and XML messages were simultaneously set to a specified size by adding dummy data to the messages. The message sizes used were: 1 kB, 10 kB, 100 kB, 1 000 kB. The majority of the measurements were repeated 100 times. However, since some of them took several hours to complete, it was necessary to reduce the number of repetitions for some of the more long running measurements, in order to stay within the time scope of the thesis work. The number of repetitions for each measured data point together with the mean- and standard deviation results has been presented. The mean recovery time was also plotted on a graph.

Note that, due to the simplifications made on the E-service model, the recovery time was measured for equal amounts JMS- and XML messages. For instance, replaying 100 messages meant replaying 50 JMS messages and 50 XML messages. In contrast, the systems at Bolagsverket send on average 15 times more XML messages compared with JMS messages, as can be seen in chapter 3.4. This effectively meant that the recovery time measurements were not conducted in the same proportion of JMS- and XML messages as compared to that for the Bolagsverket’s systems. It is important to take this into account when analyzing the results.
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

4 Methodology

2013-12-27

**Semi-automatic replay**

The recovery time for semi-automatic replay mainly depends on how quickly officials re-handle errands. Further, since no data exists regarding how long this may take, there was no way of simulating the re-handling process in a realistic manner. Therefore, instead of simulating and measuring the recovery time, an equation for the recovery time was deduced. An approximation for the maximum recovery time at Bolagsverket was calculated, based on the assumption that the mean re-handling time will, maximally, take as long as the original handling procedure.

4.3.4 **Storage requirements**

An equation for the amount of data to be logged in order to ensure that the data in the database can be recovered a number of days back in time was deduced. The data amount corresponds to the persistent storage capacity required. The results were valid for both automatic- and semi-automatic replay since they utilize the same logging mechanism. An approximation regarding the amount of data requiring to be logged during one average day’s operation at Bolagsverket was calculated and presented.

4.3.5 **Scalability**

The scalability of the implemented logging/replay system was analyzed and evaluated theoretically. This was mainly achieved by means of reasoning in relation to how the architecture affected the scalability and how the system could be expanded by adding physical resources. The following questions were discussed:

- Can the system scale to handle increased/decreased load?
- Can the system handle Bolagsverket’s systems scaling?
- Can the logging/replay system be implemented on several systems?

4.3.6 **Implementation complexity**

The complexity of implementing the logging/replay system was analyzed and evaluated in a theoretical manner. This was achieved by means of reasoning regarding the function and complexity of each
replay approach, as well as the amount of work required to implement them. More specifically, the following topics were discussed:

- How complex are the two approaches to implement in relation to each other?
- How much work is necessary in order to implement the approaches?
- Are the approaches implemented using complex algorithms, advanced techniques or advanced technologies?

4.3.7 Integration complexity
The complexity of integrating the logging/replay system with the E-service model was analyzed in a theoretical manner. This was analyzed by considering to what extent the code in the E-service model had to be modified in order to integrate it with the logging/replay system. The main purpose of this was to analyze the complexity and amount of additional code that had to be written by developers in order to utilize the logging/replay system. The property is important since it is often not economically feasible for organizations, such as Bolagsverket, to implement solutions that require significant changes in their large and complex systems. Thus the ideal situation would, from an integration perspective, be a recovery solution that could be integrated with existing systems without having to make any modifications to the systems.

4.3.8 Maintainability
The maintainability of the implemented logging/replay system was analyzed and evaluated theoretically. This was conducted by analyzing the ease by which the functionality of the system could be extended and maintained when the system had been deployed in a production environment. More specifically, the following topics were discussed:

- Maintaining the logging- and replay systems.
- The advantage of loose coupling between the logging/replay system and the E-service model.
5 Implementation

The chapter presents the requirements and implementation of the E-service model and the proof-concept solutions. The requirements for the E-service model, derived from the analysis of Bolagsverket’s IT systems are described. The technical requirements for the proof-of-concept implementation are also presented. The implementation of the E-service is explained on a relatively high abstraction level, since these details are expected to be of modest importance for the thesis. However, the implementation, function and architecture of the proof-of-concept implementation are covered in detail.

5.1 Requirements

The chapter specifies the requirements for the E-service model and the proof-of-concept implementation.

5.1.1 E-service model

A simplified and general E-service model that mimics Bolagsverket’s E-service is required to be implemented, as stated in chapter 1.4. Based on the analysis in chapter 3 the following requirements for the model are derived.

Functional requirements:

- Customers can submit applications to register new companies by specifying name, company name and address.

- Officials can pick an errand to handle, give it a unique case number and give the new company a unique organizational number.

- Officials can see a list of registered companies.

Technical requirements:

- Customers submit applications to a customer application system via a web UI. The customer application system forwards the application asynchronously to a company management system via JMS messages and a message queue.
The company management system fetches JMS messages from the queue and saves them in a company database.

The company management system enables officials to handle errands. Officials handle errands via a web interface that communicates with a client, which in turn communicates with the company management system by mapping all user interactions from the officials to XML messages. The communication between the official and the company management system is required to be synchronous. The system should be stateless and save its information in the company database. The system should also be deterministic with respect to the messages and satisfy the PWD assumption.

5.1.2 **Proof-of-concept implementation**

The functional goals for the proof-of-concept implementation are specified in chapter 1.4. The technical requirements are specified below.

- The logging system is required to implement application-level pessimistic logging in order to log all messages received by the company management system in the E-service model.

- Oracle Flashback is required to be used to recover/restore the database to a previous point in time.

- Automatic replay is required to replay JMS- and XML messages in a manner such that the company management system re-executes the messages exactly as during the original execution.

- Semi-automatic replay is required to replay JMS messages and to manually re-handle errands in an identical manner as during the original execution of the company management system. The information from the logged XML messages should be made available through a web interface during the re-handling procedure.

**Design decisions**

Designing the proof-of-concept implementation requires making trade-offs and balancing between several properties. The subchapter describes the reason behind the main design choices.
Pessimistic logging is used since it provides a simple logging and recovery procedure. It is expected to have lower implementation complexity compared with the other logging approaches. Further, it does not suffer from orphan processes and enables an individual process to recover without the necessity to coordinate with other processes. These advantages do, however, come at the cost of high failure-free overhead. Optimistic- or causal logging would have lower failure-free overhead at the cost of higher implementation complexity. It is assessed that pessimistic logging gives the best trade-off between implementation complexity and failure-free overhead.

Application-level logging is used in the proof-of-concept implementation since it is assessed to be the most straightforward and simple solution to log messages. Application-level logging is not transparent to the programmer, which is a drawback. On the other hand, this also gives the developer total control of the logging procedure, which could be beneficial in certain situations. An alternative could be to use user-level logging but, since no sufficiently good logging libraries have been found that meet the requirements of the thesis, this was not considered. It is not reasonable or possible to consider kernel- or hardware-level logging in the thesis for several reasons. Firstly, Bolagsverket uses some proprietary software and hardware that cannot be legally modified. Further, solutions involving kernel- and hardware logging cannot be used for cross-platforms since this type of logging is expected to be specific for a certain kind of hardware or software. Thus, using this type of logging would risk the logging procedure having to be modified on each occasion that Bolagsverket changes the hardware and software in their systems.

Oracle Flashback is used to recover/restore the database to a previous point in time mainly due to its simplicity and convenience. As mentioned in the theory chapter, there are several other means of accomplishing this, each of which is expected to work in as satisfactory manner as is the case for Oracle Flashback. This is also the reason why the recovery time measurements do not include the time it takes to restore the database, as stated in chapter 4.3.3.
5.2 E-service model

The E-service model is used as a test environment for the proof-of-concept implementation. It is implemented as a simplified and general model of one of Bolagsverket’s E-services using Java EE. The model is kept general in order to represent any system that communicates asynchronously using JMS messages and synchronously using XML via HTTP. The requirements for the E-service model are specified in chapter 1.4 and chapter 5.1.1.

Figure 6 presents a high-level illustration of the E-service model. Customers can, via a web browser, submit applications to register new companies. The customer application system mimics Bolagsverket’s RFB system and handles the customer’s interaction with the model. Whenever a customer has submitted an application, the system asynchronously sends a JMS message containing the errand to the company management system. The asynchronous communication is possible by utilizing an intermediate message queue.

The company management system mimics Bolagsverket’s Unireg system and provides the core functionality of the E-service. It utilizes a database to store and retrieve data related to different companies and errands. It should be noted that it is the data within this system (more specifically the database) that the proof-of-concept solutions should recover, if, for some reason, it becomes disarranged, erroneous or inaccurate. The management system will fetch and process JMS messages from the queue if it has available resources. The retrieved errands are saved in the database, waiting to be manually handled by officials. The management system’s interface towards the officials consists of a RESTful interface based on XML messages sent over HTTP, implemented using the JAX-RS API. The company management system is deterministic with respect to the XML and JMS messages. It also satisfies the PWD assumption covered in the theory chapter. Further, the system is stateless and any necessary information is saved in the company database.

The client system mimics Bolagsverket’s “webserver” and hosts a web application that enables the officials to handle errands in a synchronous manner. The client maps the official’s interactions to XML messages which are sent/received over HTTP, to and from the company management system.
Officials manage errands manually by connecting to the client system via a web browser and by choosing an available errand from a list. Officials can handle errands concerning the registration of new companies by specifying unique case- and organization numbers, and, if necessary, modify any information in the customer’s application.

The relational schema for the company database can be found in Figure 21 in appendix A.

![Diagram of the E-service model](image)

**Figure 6: A high-level illustration of the E-service model**

### 5.3 Proof-of-concept implementation

The two alternative approaches referred to as automatic- and semi-automatic replay are implemented in a proof-of-concept manner on the E-service model. The aim of the two approaches is to recover the correct data in the E-service’s company database in case it becomes disarranged, erroneous or inaccurate. Two somewhat different approaches are used for recovery, even though they are both based on the rollback-recovery techniques presented in the theory chapter. Automatic-replay recovers data by firstly rolling back to a checkpoint prior to the data becoming erroneous, before replaying all logged determinants from that point. In contrast, semi-automatic replay recovers data by firstly rolling back to a checkpoint prior to the data becoming erroneous, before replaying only...
the logged JMS messages and enabling officials to manually re-handle errands. The two alternative approaches are implemented since it has not been clearly determined whether Bolagsverket’s Unireg system is deterministic with respect to the XML messages, as stated in chapter 3.3. Thus, the latter approach does not exploit this property.

Some important points about the proof-of-concept implementations should be noted. Both automatic- and semiautomatic replay utilize the same logging mechanism for logging determinants. The logging is performed using a receiver-based, application-level, pessimistic logging technique. Further, no checkpoints are explicitly taken. When the system requires to be rolled back to a previous checkpoint during recovery, this is achieved by restoring the database to a previous point in time (simulating a checkpoint). This is possible because the system in itself is stateless, hence it will never change state over time. The system stores everything directly in the database, or in other words, if the system and database are seen as one entity, the content of the database will be the state of the entire system. Thus, restoring the database to a previous point in time effectively means rolling back the whole system to a previous point in time, which is equivalent to rolling back to a previous checkpoint.

5.3.1 Logging system

During normal execution all nondeterministic events related to the company management system are logged using an application-level and pessimistic logging approach. The events consist of all XML and JMS messages sent to the company management system, and are logged by the system upon receipt. Figure 7 illustrates how the company management system logs messages by calling the logging system, which is responsible for serializing and saving the messages in the logging database. The database adds a system-time timestamp, as well as a unique identification number to every new tuple inserted in the database. The relational schema for the logging database can be found in Figure 22 in appendix A.
Figure 7: Logging all messages during normal execution.

Figure 8 presents the main parts of the logging system. It consists of a stateless enterprise session bean with a business interface that provides a simple means of logging messages. The bean utilizes JPA to persist the log entities to the logging database and a singleton class to enable/disable the logging mechanism.

The company management system logs a message by looking up the LoggerBean in JNDI and calling the business method named logEvent with the message as the argument. The method will create and persist a new Log instance containing the serialized message and metadata about the type of message (JMS or XML) being logged. The metadata is used during replay in order to determine the type of the serialized messages.
5.3.2 Replay system

The replay system is responsible for recovering the correct data in the database in case it becomes erroneous or inaccurate. This is achieved by restoring the database to a previous point in time, before utilizing the logged messages to recover the data. The replay system implements two different approaches for data recovery: automatic replay and semi-automatic replay.

Automatic replay aims to recover data by re-sending all logged messages in the same manner as during the original execution. The procedure must be initiated manually but apart from this, it is fully automatic.

Semi-automatic replay aims to recover data by enabling officials at Bolagsverket to manually re-handle affected errands. The system ensures that the errands will be handled in the same order as during the original handling process. The system also provides the officials with all the information required to ensure that the outcome of the re-handling process will be exactly identical to the outcome of the original handling process. The latter is very important since it is critical that information, such as organizational numbers, is not changed during replay.

The replay system is also responsible for providing a web interface that enables system maintainers (or other authorized personnel) to manage
the replay system, as well as the functionality of restoring the database to a previous point. The system maintainers merely specify a previous point in time, and the system is responsible for restoring the database to the specified point and initiating the replay procedure from the same point.

During replay the time offset between the logged messages is not taken into account. This enables the re-execution to proceed faster than the original execution. Further, the system cannot, during replay, be subjected to any messages other than those being re-sent. This means that all new inbound applications from customers and handled errands from officials must be stopped. The logging mechanism must also be turned off during replay to prevent the re-sent messages from being logged once again. This is achieved by calling the logging system.

The following subchapters will cover the replay system and its main parts in more detail.

**Restoring the database**

If the data within the company database is detected to be disarranged, erroneous or inaccurate, the database is restored to a previous valid and accurate state, before initializing the data recovery procedure. Note, however, that it is not within the scope of the thesis to consider how the actual detection of the faulty information is made. It could possibly have been detected manually or by some automatic detection mechanism.

As mentioned in the theory chapter, the database can be restored to a previous point in time, in a variety of ways. The database could, for example, be restored using a backup, or using technologies such as Oracle Flashback. The proof-of-concept implementation utilizes the latter to recover the company database to a previous point in time. It also implements a means of deleting all data within the database using standard SQL, to enable a simple method for simulating replay from the point where the database was empty. These functionalities are implemented in a helper class that is used during replay.

Figure 9 illustrates a scenario where the database is restored. The timeline represents the data in the database at different points in time. At point 1, everything works as expected and the data within the database
is valid and accurate. At point 2, an event occurs which results in the data being inaccurate or invalid from this point on. At point 3, the data is detected to be erroneous. During point 4, the database is restored back to a point in time when the data was known to be accurate (point 1). Note that the restoring procedure implies that all data (both valid and invalid) from point 1 onwards, is lost.

Figure 9: Restoring the database to a previous point in time due to erroneous data.

**Synchronizing the logging- and company database**

In order to start replaying logged messages after the company database has been restored to a previous point in time, it is necessary to know exactly at which point it is necessary to begin in the log. Thus, each logged message has to be synchronized with the company database so that it is unambiguous which message should be re-sent first.

The synchronization is performed by running both databases on the same physical machine and utilizing the system time in both databases. It should be noted that the databases are not run on the same server as the E-service model or the proof-of-concept implementation. As mentioned previously, the logging database will add a system-time timestamp to every new logged determinant, which is inserted. The company database is restored to a previous system-time, before all logged messages from this point on are replayed. This setup is necessary in order to ensure that both databases utilize the same clock. As mentioned in the theory chapter, it is impossible to perfectly synchronize the clocks across distributed systems. Even a small time difference between the two databases could have a significant impact on the replay procedure. Figure 10 illustrates this issue by means of a scenario in which the company database and the logging database reside on different servers. The company database is restored to 12 o’clock before messages are replayed from 12 o’clock in the logging database. As seen in the figure,
the difference between the server clocks will cause two messages (the small vertical lines) to be incorrectly missed.

![Diagram showing the difference between server clocks](image)

**Figure 10**: Messages in the log gets missed due to differences between the databases clocks.

### Automatic replay

Automatic replay aims to replay all logged messages in the same manner as during the original execution of the company management system. This means that the messages are re-sent and re-executed by the system in the same order as during the original execution. Figure 11 presents a high-level model of the automatic-replay procedure. A system maintainer initiates the replay process by specifying a point in time at which the company database should be restored and from which the log should be replayed. The replay system is responsible for fetching the logged serialized messages from the logging database, deserializing and replaying them in the correct order. The company management system will fetch JMS messages from the queue and receive XML messages sent over HTTP as is the case during normal execution. The management system will have no clue that the messages are re-executed, apart from the fact that the logging mechanism is turned off.

It is important to ensure that the company management system has time to process a re-sent message before the next message is sent. This effectively means that the automatic replay procedure must guarantee that the messages are not replayed too quickly, in order to ensure that the partial order of the re-executed messages is maintained. This is due to the asynchronous communication model with the company management system. For example, consider the situation where a JMS message...
is replayed, immediately followed by an XML message. It is very likely that the company management system will process the XML message first, since the XML message is synchronously re-sent to the management system, while the JMS message is re-sent asynchronously and takes a route via the intermediate message queue (which may take time). For this reason a fixed delay is added after each replayed JMS message. The length of the delay is set so that there is a sufficient amount of time for the system to process each message.

Figure 11: Automatic replay: deterministically replay all logged messages to recover the data in the company database.

Figure 12 presents a UML class diagram of the main parts of the automatic replay implementation. It consists of a stateless enterprise session bean named “AutoReplayBean”, with a business interface that provides two means of replaying the logged messages. Either all the messages are replayed or all the messages from a specified point in time are replayed. The bean utilizes JPA to fetch the determinants from the database before the content is deserialized. By utilizing the metadata saved during logging, the bean can determine whether the serialized messages are XML- or JMS messages. The bean also utilizes a helper class to restore the company database to a previous point in time as seen in the UML
Semi-automatic replay

Semi-automatic replay aims to recover the data in the database by enabling the officials at Bolagsverket to manually re-handle all the affected errands from a specified point in time. The outcome of the re-handling process should be exactly the same as the outcome of the original handling process. The main difference compared with automatic replay is that this approach requires human intervention during replay and that only JMS messages are re-sent to the company management system. This means that the time required to recover the data depends on how long it takes for officials to re-handle the errands. The logged XML messages are used to provide officials with the information required to re-handle the errands in the exact same manner as during the original handling process.
Figure 13 presents a high-level model of the semi-automatic replay procedure. As in the case of automatic replay, the procedure is initiated by a system maintainer (or other authorized personnel) by specifying a point in time at which the company database should be restored and from which the replay should be started. The replay system ensures that the logged JMS messages are re-sent to the message queue and that officials can re-handle errands in the same way as during the original execution. This is achieved by providing the officials with an extra web interface that presents the necessary information. Thus the officials will utilize two different web interfaces during re-handling. The interface towards the replay system will provide the officials with the necessary information so that they are able to re-handle the errands in the same order and with the same data as during the original handling procedure. The interface towards the company management system is exactly the same as used during the original handling procedure. The officials will, more or less, take the information provided by the replay system and submit it to the company management system, using the two web interfaces.
Figure 13: Semi-automatic replay: re-handle errands to recover the data in the company database.

Figure 14 presents the UML class diagram for the main parts of the semi-automatic replay system. It consists of a stateful enterprise session bean with a business interface that provides two alternative approaches toward semi-automatic replay. The session bean also provides the means of iterating through the logged XML messages, which contains the information from the original handling procedure. The two alternative approaches either replay the whole log or replay the log from a specified point in time. The business method next is used to re-send the appropriate JMS messages to the message queue, before retrieving the information about the next errand to be re-handled. The bean uses JPA to fetch the log entities from the logging database and a helper class to restore the database to a previous point in time, prior to replay.

The web interface towards the replay system is developed using JSF and interacts with the enterprise bean via a managed bean. The interface provides officials with the necessary information in order for them to re-
handle the errands exactly as during the original handling procedure. The information includes the order in which the errands should be re-handled, as well as the information that should be submitted for each errand.

Figure 14: UML class diagram presenting the main parts of the semi-automatic replay system.
6 Results

The chapter analyzes whether the goals set for the E-service model and the proof-of-concept implementation are met. The chapter also presents the outcome of the performance tests and the theoretical evaluation.

6.1 E-service model

The goals set for the E-service model in the introduction chapter are met. The requirements derived from the analysis of Bolagsverket’s IT systems and presented in chapter 5.1.1, have all been successfully implemented. Figure 15 illustrates how a customer can submit an application via a web UI. Figure 16 illustrates how an official can handle an errand.

Figure 15: A customer submitting an application.
6.2 Proof-of-concept implementation

The goals set for the proof-of-concept implementation in the introduction chapter are all met. The technical requirements presented in chapter 5.1.2 have been successfully implemented. Figure 17 illustrates how a system maintainer can restore the company database to a previous point in time and initiate the automatic replay procedure (no further human interaction is required). Semi-automatic replay is initiated in a similar manner. Figure 18 illustrates how semi-automatic replay provides the officials with the necessary information during the re-handling procedure. The information for the next errand, in turn, to be re-handled is presented by pressing the “proceed” button.
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket
Max Sjölinder

Figure 17: The interface toward automatic replay.

Figure 18: Semi-automatic replay provides the officials with the necessary information to re-handle the errands in the same order and with the same data as during the original handling procedure.

6.3 Performance test and theoretical evaluation

The chapter presents the results of the system evaluations, conducted as described in chapter 4.3. Some of the measurements apply to both automatic- and semi-automatic replay, while others apply only to one of the two. The results are presented in the same section if they apply to both replay approaches and under separate sections otherwise. Results that are specific for Bolagsverket are also presented under a separate section.
6.3.1 Logging overhead

Figure 19 presents the arithmetic mean of the logging overhead for different JMS and XML message sizes. The graph shows the overhead in milliseconds when one message is logged. Note that the results also include network latencies due to the logging system and the logging database residing on different servers. The measurement data, together with the standard deviation can be found in Table 4 and Table 5, located in Appendix B. The result applies to both automatic and semi-automatic replay.

![Graph showing logging overhead](image)

Figure 19: A graph presenting the overhead of logging one message for different message sizes.

6.3.2 Data accuracy after replay

As stated previously, the data accuracy- and the recovery time measurements were conducted simultaneously. All of these measurements resulted in accurate data after replay.
6.3.3 Recovery time

The subchapter presents the result regarding the time it takes to recover the data in the database (sometimes referred to as replay time) for both replay approaches. Note that the measurements and equations presented do not include the time it takes to restore the database.

Automatic replay

Figure 20 presents the arithmetic mean of the recovery times for different numbers of replayed messages and message sizes. The number of replayed messages is comprised of equal amounts of JMS and XML messages, e.g. replaying 10 000 messages implies replaying 5 000 XML messages and 5 000 JMS messages. Note that the results also include network latencies since the replay system and the logging database reside on different servers. Further, the results include a fixed delay (the sending process sleeps) after each re-sent JMS message. The length of the delays increases as the message size increases, since more processing time is required. The lengths of the delays used are: 0.5 ms, 5 ms, 50 ms, 500 ms. The delays correspond to JMS messages with ascending message size, starting with the 1 kB message size. The measurement data, in addition to the standard deviation and number of repeated measurements for each data point can be found in Table 6, Table 7 and Table 8, respectively, located in Appendix B.
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

6 Results
2013-12-27

Figure 20: A graph presenting the recovery times for automatic replay.

Automatic replay with respect to Bolagsverket’s systems

By utilizing the statistics presented in chapter 3.4, the total number of messages to be replayed in order to recover an average day’s work at Bolagsverket is calculated to be approximately 30 770 messages. Further, the size of the messages within Bolagsverket’s system is on average 100 kilobyte. If this information is transferred to the E-service model, it can be seen that the recovery time is approximately 70 minutes. It is important to note that this approximation does not take into account the proportion of JMS- and XML messages, as described in chapter 4.3.3.

Semi-automatic replay

The replay time for semi-automatic replay depends mainly on how long it takes for officials at Bolagsverket to re-handle each errand and the number of working officials. Equation (1) can be used to calculate the replay time when the number of errands to re-handle is \( n_i \) and the number of JMS messages to re-send is \( m \). The variable \( t_i \) denotes the re-handling time for errand \( i \), and \( k_i \) denotes the time to re-send JMS mes-
sage $j$. The number of officials working with the re-handling of errands is denoted by $|\text{officials}|$.

It may be neither realistic nor convenient to be required to know each individual re-handling time and the replay time for each individual JMS message in order to approximate the replay time. An approximation of the replay time can be calculated by utilizing the arithmetic mean $\bar{t}$ of the re-handling time and ignoring the replay time for the JMS messages, as seen in Equation (2). The approximation can provide valuable information, especially if the re-handling time is significantly larger than the JMS replay time, which it is expected to be for the majority of the time.

\[
\text{replay time} = \frac{\sum_{i=1}^{n} t_i + \sum_{j=1}^{m} k_j}{|\text{officials}|}
\]  

(1)

\[
\text{replay time} \approx \frac{\sum_{i=1}^{n} \bar{t}_i}{|\text{officials}|} = \frac{\bar{t}n}{|\text{officials}|}
\]  

(2)

**Semi-automatic replay with respect to Bolagsverket’s systems**

Even though no real world data exists regarding the average re-handling time at Bolagsverket, an expected maximum replay time can be calculated by utilizing the statistics from Bolagsverket presented in chapter 3.4. It is expected that the re-handling of an errand will, maximally, take as long as the original handling procedure, since the officials have access to the result of the original procedure and it is therefore not necessary for them to redo all work. If the average re-handle time is taken to be the average handling time during normal operations, an approximation of the maximum replay time for a whole workday (8 hours) can be calculated using Equation (2), as follows:

\[
\text{replay time} \approx \frac{0.92 \times 1923}{220} \approx 8 \text{ hours}
\]

The calculation also assumes that all available officials work with re-handling errands during the recovery procedure. The result is expected since the time to re-handle a whole workday should take 8 hours (a workday) as this is the amount of time required to handle and re-handle an errand.
6.3.4 Storage requirements

Equation (3) presents a function to calculate the amount of data that must be stored in order to ensure that the data in the database can be recovered from an event occurring $x$ days back in time. The variables $n_{\text{jms}}$ and $n_{\text{xml}}$ denote the number of JMS and XML messages that pass through the E-service during a particular day. The size of each JMS and XML message is denoted by the variables $s_{\text{jms}}$ and $s_{\text{xml}}$, respectively. The variables $m_{\text{jms}}$ and $m_{\text{xml}}$ denote the size of the extra metadata saved for each logged JMS and XML message. An approximation of the quantity can be calculated by utilizing the arithmetic mean of the values as seen in Equation (4). The size of the metadata added to each JMS- and XML message is 44 bytes.

\[
f(x) = \sum_{d=1}^{x} \left( \sum_{i=1}^{n_{\text{jms}}} (s_{\text{jms},i} + m_{\text{jms},i}) + \sum_{j=1}^{n_{\text{xml}}} (s_{\text{xml},j} + m_{\text{xml},j}) \right)
\]

\[
f(x) \approx x \left( \bar{n}_{\text{jms}} (\bar{s}_{\text{jms}} + \bar{m}_{\text{jms}}) + \bar{n}_{\text{xml}} (\bar{s}_{\text{xml}} + \bar{m}_{\text{xml}}) \right)
\]

Storage requirements with respect to Bolagsverket’s systems

An approximation of the amount of data to be stored by Bolagsverket in order to enable recovery of $x$ days back in time can be calculated by utilizing Equation (4) and the statistics presented in chapter 3.4. The approximation is calculated in the following manner:

\[
g(x) \approx x \left( 1923(10^5 + 44) + 1923 \times 15(10^5 + 44) \right) \approx 3.08 \times 10^9 x
\]

Thus, logging one average day’s operation would yield, approximately, 3 gigabytes of data, as the following calculation suggests:

\[
g(1) \approx 3.08 \times 10^9 \text{ bytes}
\]

6.3.5 Scalability

Bolagsverket achieves scalability in their systems by load balancing between several nodes that are running the same software. Thus, by adding or removing hardware resources it is possible to scale the systems as required. A natural aspect to consider is whether the logging/replay system can cope with multiple nodes. This is especially important for the logging system which is intended to be used during
failure-free operations. The replay system does not have the same requirements with regards to scalability since it is expected to be used very rarely and when it is used, it replays one message at the time. The logging system is expected to handle multiple nodes that simultaneously log the received messages. The logging system will, however, treat all logged determinants as if they have all come from the same source. This effectively means that it is not possible to distinguish which of the nodes logged a particular determinant. This can have consequences during replay in the case for which it is required that a determinant is replayed to exactly the same node that originally logged it. In most cases this is not expected to be a problem since the nodes are, essentially, copies of each other and provide the exact same functionality. Nevertheless, if this extra functionality is indeed required, the current logging and replay system is expected to be easily extended for the purpose.

Another aspect to consider is whether the logging mechanism itself is scalable, i.e. whether it is possible to add or remove hardware resources in order to scale the logging system. It is expected that the logging system should be able to achieve scalability by load balancing between several nodes running the same software, in the same manner as in Bolagsverket’s systems. However, before this can be achieved an identified problem involving the use of a singleton class in the logging system has to be resolved. The singleton class is used to enable/disable the logging mechanism for a single node. This appears to be problematic when multiple nodes are used, since the logging mechanism should be enabled/disabled as one combined entity across the different nodes. The issue could be resolved by setting a flag in the database, indicating whether the logging is enabled or not, rather than handling this on a per-node basis.

The main concern regarding the scalability of the logging/replay system is the design choice to run both the logging- and the company database on the same physical server. The reason for this was to ensure that both databases could be synchronized by utilizing the same clock, as described in chapter 5.3.2. Two problems have been identified due to this decision. Firstly, the database could be a bottleneck in the system if there is a heavy load on the database server. This could, possibly, be resolved by upgrading the physical resources such as CPU, or by using techniques such as database partitioning [51]. Secondly, it is likely that the
logging/replay functionality will be added to more systems in which databases are run on different database servers. This effectively means that logging databases have to be spread out over every database server that contains a database, which should be recoverable in case of failures. A better solution would be to gather all logged determinants from every system, in separate tables on a single database server. This could be achieved by developing a solution that does not depend on the system-time but still synchronizes the databases.

6.3.6 Implementation complexity

The complexity of implementing automatic replay versus the complexity of implementing semi-automatic replay was relatively equal. Both recovery methods use a similar approach during replay. Hence, they both restore the database to a previous point in time, before fetching and deserializing the determinants from the database.

In the case of automatic replay, the JMS- and XML messages are re-sent in the original order. This was relatively straightforward to implement. However, it should be noted that automatic replay must implement a means of ensuring that the receiver of subsequent messages have had time to process every received message before a new message is sent, as described in chapter 5.3.2. This was conducted by adding a fixed delay after each re-sent JMS message, even though more reliable and sophisticated means of accomplishing this might exist.

In the case of semi-automatic replay, only JMS messages are re-sent in order not to exploit the unknown deterministic property of the XML interface. The errands are instead re-handled by officials in the exact same manner, providing the exact same information as during the original handling procedure. This required an extra interface since officials were required to have access to the logged XML messages containing the information from the original handling procedure. Due to the extra web interface, implementing semi-automatic replay required more work compared to that for implementing automatic replay. Implementing semi-automatic replay at Bolagsverket is expected to require substantial work, since the information saved in the XML message has to be presented to the officials in an appropriate manner. Further, it should be noted that registering new companies is only one of Bolagsverket’s responsibilities. This effectively means that the web
interface may have to be implemented in such way that different information is presented for different kinds of re-handled errands.

Neither automatic- nor semi-automatic replay is implemented using complex algorithms, advanced techniques or advanced technologies. Thus, implementing the approaches is considered to be a relatively straightforward task, even though they obviously require different amounts of work.

6.3.7 Integration complexity

The logging system utilizes application-level pessimistic logging. The logging system was required to be integrated with the E-service model in order to log all incoming messages. This was performed by adding code to the two parts of the E-service that handle incoming JMS- and XML messages. The added code first looks up the logging bean in JNDI (this is done once in the constructor), before logging each message using the following code snippet:

```java
if(loggerBean.getIsLoggingActive()){
    loggerBean.logEvent(message);
}
```

No other modifications to the E-service model are made. It should be noted that the logging system does not explicitly take checkpoints. Whenever a checkpoint is required, this is provided by the database by simply restoring it to a previous state. Thus, the implicit checkpointing procedure is totally transparent to the developer. Further, integrating the replay system with the E-service does not require any modifications to the E-service’s code.

6.3.8 Maintainability

The functionality of logging and the functionality of replaying are separated in two distinct systems. The systems do not have anything in common, with the exception of the logging database that contains all logged determinants. There is a key advantage with this design from a maintenance perspective. The functionality of either system could be modified or extended without affecting the other, as long as the logged determinants are not affected by the change.
Further, the logging- and replay system is loosely coupled with the E-service model which eases maintenance for both the E-service and the logging/replay system. The whole logging system could, for instance, be replaced with another solution without having to modify the E-service model, as long as the business interface is the same. This provides a great deal of flexibility.
7 Conclusions

The chapter covers the outcome and goals of the thesis. This includes unexpected problems, identified limitations with the two recovery approaches, and ways in which the methodology could be improved to increase the quality and generality of the results. The chapter further discusses the results and suitability of the two alternative recovery approaches with respect to Bolagsverket. A list of recommendations is also provided. Finally, the chapter covers societal- and ethical aspects of the thesis, as well as future work.

7.1 Evaluating goals and outcome

The thesis has met all the goals set in the introduction chapter. The main goal was to investigate whether automatic- and semi-automatic replay could be used by Bolagsverket to recover the correct data in a database that contains inaccurate or erroneous information due to past faults. Another important goal was to produce general results that could be of value for other organizations with similar problems. The two goals were achieved by implementing both replay approaches on a simplified and general model of Bolagsverket’s RFB E-service. The requirements for the model were derived from the analysis of Bolagsverket’s systems. Research was also conducted that covered fault tolerance in distributed systems, especially with respect to rollback-recovery.

The outcome of the performance test and the theoretical evaluation shows that both approaches successfully recover the data within the database belonging to the company management system. The results are general to any stateless system for which a database should be recoverable and which communicates asynchronously using JMS messages and synchronously using XML sent over HTTP.

The main newsworthiness and contribution of this thesis is considered to be the fact that the study is made on a real case scenario. Very little research has been discovered that takes real business- and organizational needs as a starting point for developing a rollback-recovery solution that enables the recovery of faulty data within a database. Further, no research has been discovered that utilizes a database’s own checkpoint-
and rollback facilities in order to rollback a system’s state, before replaying the logged determinants and thereby also recovering the correct data within the database.

7.2 Results and experiences
The subchapter discusses important aspects of the results. The chapter also discusses some of the problems experienced during the thesis work.

7.2.1 Evaluating results
The key result from the performance test and the theoretical evaluation is considered to be the fact that both automatic- and semi-automatic replay managed to accurately recover the correct data after the database, for some reason, contained erroneous or inaccurate information. The result is a prerequisite for both replay approaches. The results from the logging overhead measurements suggest that the pessimistic logging operation has linear performance. Thus, a doubling of the message size yields, approximately, a doubling in overhead time. It is expected that the serialization of the messages and the insert operation in the database are the main contributors to the overhead time and that this also is the main reason why the resulting graph is so smooth. The operations of serialization and database insert are expected to be stable operations in which execution times do not significantly vary between subsequent runs. It should also be noted that default Java serialization is considered to be a relatively slow operation [52][53].

The recovery time measurements for automatic replay suggest that increasing the number of replayed messages by a factor of ten also results in the recovery time taking ten times longer. This is expected since equal sized messages should take, approximately equal times to fetch and replay. A ten times increase in the number of messages should therefore result in an approximate ten times increase in recovery time. It is interesting to note that increasing the message size by a factor of ten, only yields an approximately seven times increase in recovery time. This is probably due to the database’s retrieval performance. It should also be noted that the recovery time depends on the complexity and amount of processing that has to be performed by the system re-executing the replayed messages. Thus, a more complex system would, most certainly, have longer recovery times compared with the times measured for the simple company management system. It was not expected that the
recovery time measurements would result in such high values for some of the data points. For example, it takes almost 9 hours to replay 30 000 messages when each message has a size of 1 000 kB. However, by analyzing this more closely it can be seen that the result is, in fact, very reasonable. By dividing the recovery time by the number of replayed messages, the average replay time per message is calculated to be approximately one second/message. It was determined that the average time to log a 1 000 kB message also results in approximately one second per message, which is reasonable. In the case of semi-automatic replay, it is expected that the re-handling time is the main contributor to the recovery time and that the time to resend a JMS message is negligible in comparison.

The theoretical evaluation of the storage requirements resulted in an equation for the amount of data that must be stored in order to ensure that the data in the database can be recovered for a given number of days back in time. The equation suggests that the size of each logged determinant consists of 44 bytes of metadata, plus the size of the logged message. The cost of saving the extra 44 bytes is considered to be low. It should, however, be possible to decrease the size of the metadata by choosing the format of the saved metadata more wisely.

The scalability evaluation identified a problem involving the use of a singleton class in the logging system. It is expected that the issue can be resolved by utilizing a database. By fixing this problem, the logging system should scale by adding physical resources and by load balancing between multiple nodes. The logging system is also expected to handle a target system (the E-service in this thesis) which is itself scaled by load balancing between multiple nodes. The theoretical evaluation also identified problems with the design decision with regards to running both the logging- and company database on the same physical machine. This means that every database that should be made recoverable by the methods described in this thesis, must be run on the same machine as the log itself. This could have both performance and scalability problems. The issue could be resolved if the two databases were synchronized without using the system time. This is, however, not easily achieved since the company database should be restored to a previous point in time and the logged determinant from this point on should be replayed.
The evaluation of the implementation complexity with regards to automatic- or semi-automatic replay shows that neither approach is difficult or complex to implement. The design decision to use pessimistic logging is considered to be the main reason behind this result.

The logging system utilizes application-level logging, which requires the developer to manually write the code that logs the incoming messages in the target system. The ideal situation would, obviously, be a transparent logging solution that relieves the developer of this responsibility. This would also remove the risk of human-made faults, which for example could be caused by a developer forgetting to add the necessary code. Despite this drawback, the complexity of integrating the logging/replay system with the target system is very low and the amount of code to be written is minimal. The developer is merely required to add code at the interfaces receiving the messages. Further, the checkpointing is transparent to the developer since this is handled by the database.

The logging- and replay systems are easily maintained. This is mainly due to the two functionalities being separated into two distinct systems. It is also due to the loose coupling between the logging/replay system and the target system. The separation of functionality makes it possible to implement logging without having to implement replay. This could be beneficial in a situation where it is urgent to start logging incoming messages for fault tolerant purposes, while still having the possibility to implement replay at a later stage.

7.2.2 Improving the quality of the performance test

As a consequence of utilizing the simplified E-service model, the recovery time measurements for automatic replay were conducted on equal amounts of JMS- and XML messages. More general results can be obtained by also conducting the measurements on different proportions of JMS- and XML messages. This can be accomplished by redesigning the E-service model or by simply resending the same message multiple times. It would also be interesting to obtain results regarding the individual recovery times for the JMS and XML messages. The thesis has currently only measured the recovery times by considering the two message types collectively.
Running the databases and message queue on the same machine as the E-service model and proof-of-concept implementation would remove the effects of network fluctuations and latencies. This would, on the other hand, also decrease the degree to which the setup of the E-service model resembles the systems at Bolagsverket. It may also be possible to improve the measurements by separating the load testing and the E-service model on different machines. Currently, JMeter is run on the same machine as the deployed Java EE applications, which might affect the results.

7.2.3 Unexpected problems

It was not anticipated beforehand that it would be required for the automatic replay procedure not to replay the logged JMS messages too quickly in order to ensure that the order of the re-processed messages would be maintained. Before this was identified, the automatic replay procedure resulted in some strange and confusing outcomes. In the majority of cases the procedure would only partially recover the correct data, but once in a while it actually did manage to recover everything. It was discovered that when both a XML- and JMS message were replayed within a short time interval, the XML message would, generally, be processed first, even though the JMS message was sent first. This was due to the JMS message being sent asynchronously and therefore it proceeded along the more time consuming path via the message queue. This effectively meant that some of the replayed XML messages would attempt to “re-handle” an errand that did not yet exist. The issue was solved by adding a fixed delay after each replayed JMS message, which unfortunately had a negative impact on the recovery time. It is, however, expected that much better and reliable ways of solving the issue do exist. The use of a fixed delay is regarded as a relatively bad and vulnerable solution since, in the case in which the delay is set at too small a value, the situation described above will occur. On the other hand, if the delay is set to be too large there will be a big performance loss. The development of a better solution is regarded as future work.

The Oracle Flashback technology stores backup- and recovery related files in a flash recovery area [54]. The size of this area was initially too small to be able to hold the information necessary to restore the company database one day back in time. This issue was solved by allowing a
7.3 Limitations with the proof-of-concept implementation

Note that this subchapter only presents the identified limitations that have not been covered in other parts of the thesis.

A limitation has been identified that concerns the situations from which automatic- and semi-automatic replay are able to recover data. If the fault causing the inaccurate or erroneous information within the database originates from the XML- or JMS messages itself, the error will only be repeated during replay. Thus, the approaches studied in the thesis are unable to recover from faults originating from the messages. It is therefore crucial to thoroughly validate the messages prior to sending them in order to reduce the risk of logging erroneous messages. The two approaches are able to successfully recover the data in situations where the fault leading up to the error or failure, has modified the content of the database directly. However, any perfectly valid changes made directly to the database by, for example, a DBA, cannot be replayed during recovery. The DBA would, for instance, connect and work with the database directly which would not be reflected in the log.

During the recovery procedure, the company management system cannot be subjected to any messages other than those being replayed. As a consequence, any users that directly- or indirectly utilizes the system will be affected, which of course is a drawback.

7.4 Suitability for Bolagsverket

From the result of the proof-of-concept implementation the assessment is that both replay approaches could be feasible options for Bolagsverket, provided that certain conditions are met and depending upon the situation. It is assessed that semi-automatic replay is a suitable option if the Unireg system is proven to be nondeterministic with respect to the XML messages, or in the case that this property cannot be determined. In contrast, the assessment is that automatic replay is a suitable option if it is proven that the Unireg system is deterministic with respect to the XML messages and that the PWD assumption holds. It should be noted that semi-automatic replay would also successfully handle the latter situation, but automatic replay is considered to be a

DBA at Bolagsverket to assign more space to the flash recovery area of the company database.
better alternative due to its superior performance. However, it is expected that the implementation of automatic replay will be complicated by the large number of interacting distributed systems at Bolagsverket. Some of these systems have complex dependencies that might have to be taken into account during automatic replay. Semi-automatic replay is not expected to have this disadvantage.

Bolagsverket’s Unireg system is considered to be immensely more complex as compared to the implemented E-service model. Therefore, it is expected that the recovery time for the Unireg system will be higher than the recovery time measured for automatic replay on the E-service model, since it is expected that each replayed message will require more processing time by Unireg. Nevertheless, it is expected that the result from the E-service model should provide a rough estimate of the recovery time and thereby provide valuable information. The other evaluated properties do not depend on the complexity of the target system to the same extent and should provide reasonably accurate estimates for the systems at Bolagsverket.

The overhead per logged message was measured to be approximately 100 milliseconds. It is not within the scope of the thesis to determine whether this failure-free overhead per logged message is viable from Bolagsverket’s perspective. There are several aspects that have to be taken into consideration here, many of which are non-technical. It should, for example, be determined to what extent the logging degrades Unireg’s performance and whether the users can tolerate this.

The storage space that must be provided by Bolagsverket in order to enable recovery one day back in time was calculated as being 3 gigabyte. It is, however, not within the scope of the thesis to determine how far back in time the data should be recoverable. The cost of storage must be weighed against the cost of losing data.

The recovery time for one average day’s operation was measured to be approximately 70 minutes for automatic replay. This result should, however, be seen as a rough estimate since Unireg is expected to require longer processing times as has been previously mentioned. In the case of semi-automatic replay, the recovery time for one average day was calculated at a maximum of 8 hours. The calculation assumes that the average time it takes to re-handle an errand will, maximally, take as
long as handling an average errand during normal operation. The calculation also assumes that all available officials will be working with the re-handling of errands during the recovery procedure.

Both of these recovery times should be related back to the event that occurred at Bolagsverket approximately ten years ago, which was presented in the introduction chapter. It should be recalled that the event required around 20-30 employees, each working for roughly 15-30 hours, before the data could be recovered by manually tracing system- and database logs. Consideration should also be given to the fact that this was due to a fault that was discovered almost immediately. Since this event, Bolagsverket’s systems have grown and become significantly more complex which has probably also increased the time it would take to recover data. Therefore the approach used ten years ago should not be relied upon to solve a similar problem occurring today. Thus, it is clear that both recovery approaches discussed and evaluated in this thesis can be of great value to Bolagsverket. Even though neither automatic- nor semi-automatic replay may be able to guarantee that the data can be recovered, they do take a more systematic and preventive approach toward recovery. This should at least reduce the risk of losing data, which may be of greater importance than any performance characteristics.

However, before the recovery approaches are implemented in a production environment, it is necessary to research the effect of nondeterministic events such as I/O interrupts. This falls outside the scope of the thesis and is therefore regarded as future work.

7.4.1 Recommendation
The result of the thesis is summarized into the following recommendation to Bolagsverket. The list is in order of descending priority.

1. Seriously consider starting to log messages. This will at least ensure that the execution of Unireg can be tracked and that the information processed by the system is not lost. The cost of implementing this is expected to be approximately 100 ms overhead per message and a 3 GB/day storage requirement.

2. Determine whether Unireg is deterministic with respect to the XML messages and whether the PWD assumption holds.
3. Depending on the result from point 2, implement a suitable replay approach in Bolagsverket’s test environment and evaluate the performance, especially with regards to nondeterministic events such as I/O interrupts and clock events.

7.5 Societal aspects

There are significant benefits in enabling the recovery of erroneous, lost or inaccurate data in Bolagsverket’s databases. Not only does this benefit Bolagsverket as an organization, but also Swedish companies and society. There could be numerous possible consequences due to erroneous, lost or inaccurate data within Bolagsverket’s databases. Companies could, for example, be incorrectly placed into liquidation, which is the juridical process of bringing a company to an end. It is very likely that a liquidated company will have difficulties during its normal business operation due to problems with banks, etc. Bolagsverket may also sell or base its decisions on faulty data. Obviously, this could have economic, juridical and reputational implications for the involved parties, which in turn also affect society. The societal importance for an authority such as Bolagsverket to be able to recover from unforeseen events that result in erroneous, lost or inaccurate data should therefore be clear. This is particularly true as it appears that it is the citizen’s opinion that among the most important e-Government (electronic government) qualities are information reliability, system accuracy and system recoverability [55].

7.6 Ethical aspects

The logging mechanism will log all of a system’s received messages for recovery purposes. However, the large amount of data stored within the log could be used in ways other than originally intended, which could possibly intrude on the user’s privacy. Messages are often sent as a result of a system’s interaction with the users and contain information regarding these interactions. Further, a timestamp is added to every logged message. This effectively means that the log contains information regarding when and what a user has been doing in the system. In some cases, it is also expected that the messages contains information regarding from where the user has interacted with the system. The information could possibly be used to oversee and monitor users in the system and thereby intrude on their privacy. It can be argued that much of the information contained within the log is already saved in Bolagsverket’s
systems. This may very well be true, but it is expected that this information is spread out in different formats, across multiple tables and databases. The log gathers this information in one place and in the same format, which makes it ideal for analysis and data mining. It is the ethical responsibility of the engineers and developers to ensure that the privacy of the users is upheld.

7.7 Future work

7.7.1 Improving performance

When constructing the logging/replay system several different properties have to be balanced against each other and trade-offs between the properties are required to be made. It is important to take this into account when attempting to improve the performance of the system. It would be very interesting to investigate different ways of enhancing the performance of the logging/replay system. Some changes might be able to improve the performance without affecting other properties, while other changes might affect the properties in an unacceptable manner.

The design choice with regards to utilizing pessimistic logging results in the company management system having to wait for a message to be serialized and inserted into the database, before it can proceed. By utilizing another logging approach, such as causal logging, it is expected that the logging overhead will significantly decrease. This would, however, come at the cost of higher implementation complexity. Further, performance gains are also expected in both the logging- and replay system by replacing Java’s default serialization with other techniques. It would, for example, be interesting to investigate the performance gains of utilizing techniques such as Google protocol buffers [56], or serialization libraries such as Kryo [57]. Another approach could be to investigate the possibility of saving the logged messages in clear text which would eliminate the need for serializing the XML messages. It should also be possible to tune the logging database to gain performance. The replay system currently fetches and processes one logged determinant at a time, which can be ineffective. A more effective approach would be to retrieve a “batch” of logged determinants and process each individually. Note however that caution is required in order not to fetch too many determinants at once, which would risk running out of memory.
As mentioned earlier, due to the asynchronous communication model between the replay system and the company management system, a fixed delay is added after each replayed JMS message to maintain the order of the re-processed messages. This also reduces the performance of the system since the delay has to be sufficiently large to ensure that every message has time to be re-processed within the delay period. It is therefore expected that the recovery time can be reduced by replaying the JMS messages synchronously to the company management system. However, a problem with this approach has been identified and must be investigated further. It is not clear how to re-send the JMS messages synchronously without having to modify existing code in the company management system. It has already been stated that the company management system listens to the message queue and fetches messages from the queue when resources become available.

An alternative solution may be to leave the asynchronous communication model as it is and instead ensure that the replay system does not replay any XML messages before it knows that all replayed JMS messages have been processed. This can be accomplished by allowing the replay system to listen to the message queue and only replay XML messages when there are no JMS messages on the queue.

The use of a fixed delay to maintain the order of the replayed messages is regarded as a relatively poor and vulnerable solution. The two approaches discussed above do not rely on such delay and thus are expected to be much better and robust solutions.

### 7.7.2 Other nondeterministic events

The thesis has not considered nondeterministic events such as I/O interrupts, power outages or clock events. These are all highly relevant considerations that should be investigated before the approaches presented in this thesis can be adopted in a production environment. Research should be conducted that investigates whether the log can be maintained in an accurate state in case nondeterministic events such as IO exceptions occur during a systems normal execution. Further, it should also be investigated whether the recovery procedure can be carried out properly if IO exceptions occur.
7.7.3 Undo and Redo

It would also be very interesting to investigate how the log could be used to achieve “undo and redo” capabilities similar to Joyce and 3R, presented in the chapter 2.6.3. The log could be seen as a re-executable timeline of a system’s original execution. It would therefore be interesting to investigate how the timeline could be modified before replay and how this would affect the outcome. It may, for example, be possible to modify the data within the logged messages, change the order in which the messages are replayed, or merely omit some of the logged messages during replay. This could possibly also be used to enable recovery from faults and errors that originate from the messages itself, which the current replay system is unable to handle. Thus, if it is known which message contains an error or fault, it may be possible to fix the problem by modifying the particular logged message prior to replay. Undo and redo capabilities would essentially provide Bolagsverket, as well as other organizations, with a convenient way to “travel in time” in order to fix past problems.
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

References


Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket
Max Sjölinder


78
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket
Max Sjölinder


References 2013-12-27
Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

References

2013-12-27


Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

References

2013-12-27


Preventing data loss using rollback-recovery - A proof-of-concept study at Bolagsverket

Max Sjölinder

References

2013-12-27


Appendix A: Relational schema for the databases

The relational schema for the company- and logging database can be seen in Figure 21 and Figure 22 respectively.

**REGISTER_COMPANY_APPLICATION**

| ApplicationID | CompanyName | CompanyRegistrarName | CompanyRegistrarAddress | DateSubmitted |

**COMPANY_DIRECTORY**

| CompanyOrgNr | CompanyName | CompanyRegistrarName | CompanyRegistrarAddress | EstablishmentDate |

**HANDED_CASE_DIRECTORY**

| CaseNr | Year | CompanyOrgNr |

Figure 21: Relational schema for the company database.

**LOG**

| LogID | LogType | LogTime | LogContent |

Figure 22: Relational schema for the logging database.
Appendix B: Measurement data

The appendix contains a summary of the data collected during the performance tests.

Logging overhead

Table 4 contains the arithmetic mean of the logging overhead when logging one message, for different message sizes. Table 5 contains the standard deviation for the logging overhead measurements.

Table 4: The arithmetic mean of the logging overhead for different JMS- and XML message sizes.

<table>
<thead>
<tr>
<th>Message size (kB)</th>
<th>Overhead time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMS message</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>50</td>
<td>58</td>
</tr>
<tr>
<td>100</td>
<td>108</td>
</tr>
<tr>
<td>250</td>
<td>251</td>
</tr>
<tr>
<td>500</td>
<td>489</td>
</tr>
<tr>
<td>750</td>
<td>750</td>
</tr>
<tr>
<td>1 000</td>
<td>1012</td>
</tr>
</tbody>
</table>

Table 5: The standard deviation for the logging overhead.

<table>
<thead>
<tr>
<th>Message size (kB)</th>
<th>Standard deviation (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JMS message</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
</tr>
<tr>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td>250</td>
<td>48</td>
</tr>
<tr>
<td>500</td>
<td>39</td>
</tr>
<tr>
<td>750</td>
<td>81</td>
</tr>
<tr>
<td>1 000</td>
<td>114</td>
</tr>
</tbody>
</table>
Recovery time for automatic replay

Table 6 contains the arithmetic mean of the recovery times for automatic replay, using different message sizes and different numbers of replayed messages. The number of replayed messages consist of equal amounts of JMS and XML messages, e.g. replaying 10 000 messages implies replaying 5 000 XML messages and 5 000 JMS messages. Table 7 and Table 8 present the standard deviation and the number of times the measurements for each data point was repeated.

Table 6: The arithmetic mean of the recovery times for automatic replay.

<table>
<thead>
<tr>
<th>Number of replayed messages</th>
<th>Replay time for different message sizes (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 kB</td>
</tr>
<tr>
<td>100</td>
<td>0.004</td>
</tr>
<tr>
<td>1 000</td>
<td>0.047</td>
</tr>
<tr>
<td>10 000</td>
<td>0.493</td>
</tr>
<tr>
<td>30 000</td>
<td>1.414</td>
</tr>
</tbody>
</table>

Table 7: The standard deviation for the recovery times.

<table>
<thead>
<tr>
<th>Number of replayed messages</th>
<th>Standard deviation (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 kB</td>
</tr>
<tr>
<td>100</td>
<td>0.001</td>
</tr>
<tr>
<td>1 000</td>
<td>0.007</td>
</tr>
<tr>
<td>10 000</td>
<td>0.084</td>
</tr>
<tr>
<td>30 000</td>
<td>0.074</td>
</tr>
</tbody>
</table>

Table 8: The number of times the measurements were repeated.

<table>
<thead>
<tr>
<th>Number of replayed messages</th>
<th>Number of repetitions for different message sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 kB</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1 000</td>
<td>100</td>
</tr>
<tr>
<td>10 000</td>
<td>100</td>
</tr>
<tr>
<td>30 000</td>
<td>100</td>
</tr>
</tbody>
</table>