New method for data replication in distributed heterogeneous database systems

Miroslaw Kasper
Department of Computer Science
AGH – University of Science and Technology

Supervisor: Grzegorz Dobrowolski

Krakow, 2011
I dedicate this thesis to my wife Iwona
Acknowledgements

I would like to acknowledge my supervisor Grzegorz Dobrowolski for his tireless guidance and many useful comments about the material in the thesis. His intuition, insight and ideas were crucial in helping me progress in this line of research, and more importantly, in obtaining useful results.
## Contents

1 Introduction

2 Database replication
   2.1 Database replication definition
   2.2 Replication Model
   2.3 Development of replication techniques
   2.4 Review of the approaches
      2.4.1 Time of transactions
         2.4.1.1 Eager replication
         2.4.1.2 Lazy replication
      2.4.2 System architecture
      2.4.3 Middleware based replication
      2.4.4 Server interaction
      2.4.5 Transaction termination
      2.4.6 Environment complexity
   2.5 Review of replication techniques usage
      2.5.1 Transactional data processing
      2.5.2 Large amount of data
      2.5.3 Multimedia data
      2.5.4 Data in mobile systems
      2.5.5 Real time data processing
      2.5.6 Spatial data storing
      2.5.7 Replication for High Availability
   2.6 Summary


# CONTENTS

3 Theta replication method

3.1 General assumptions and requirements .......................... 50
3.2 Theta replication architecture ................................. 52
3.3 Approach details .............................................. 55
  3.3.1 Middleware components .................................. 55
  3.3.2 Transaction processing .................................. 57
  3.3.3 Communication in middleware layer ..................... 60
  3.3.4 Conflict resolution ...................................... 62
  3.3.5 Failure resistance ....................................... 68
3.4 Complexity analysis ........................................... 70
  3.4.1 Selection of the transaction identifier .................. 71
  3.4.2 Conflict resolution ...................................... 71
3.5 Summary ..................................................... 72

4 In-laboratory testing ............................................. 74

4.1 Testing implementation ........................................ 76
4.2 Performance indices and testing plan ......................... 78
4.3 Results ...................................................... 85
  4.3.1 Scalability ............................................... 86
  4.3.2 Percentage of reads ..................................... 92
  4.3.3 Conflicting transactions ratio .......................... 94
  4.3.4 Size of database copies ................................ 96
  4.3.5 Portability ............................................. 100
4.4 Summary ..................................................... 100

5 Real life evaluation ............................................... 102

5.1 Introduction to IBIS system .................................. 102
  5.1.1 System logical structure and architecture ............... 103
  5.1.2 Agents interactions ..................................... 108
5.2 Database architecture ........................................ 109
5.3 Replication implementation for IBIS ......................... 114
  5.3.1 Requirements ........................................... 115
  5.3.2 Data replication architecture .......................... 116
  5.3.3 Data flow and data management in replicated IBIS ..... 119
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.4 Implementation details</td>
<td>122</td>
</tr>
<tr>
<td>5.4 Evaluation scenarios and results</td>
<td>124</td>
</tr>
<tr>
<td>5.4.1 Failure resistance</td>
<td>125</td>
</tr>
<tr>
<td>5.4.2 Web Interface</td>
<td>129</td>
</tr>
<tr>
<td>5.4.3 Crawling</td>
<td>134</td>
</tr>
<tr>
<td>5.4.4 Data Searching</td>
<td>139</td>
</tr>
<tr>
<td>5.4.5 Strategy management</td>
<td>144</td>
</tr>
<tr>
<td>5.4.6 Method adaptation</td>
<td>149</td>
</tr>
<tr>
<td>5.5 Summary</td>
<td>150</td>
</tr>
<tr>
<td>6 Conclusions</td>
<td>152</td>
</tr>
<tr>
<td>A Theta method software deployment</td>
<td>155</td>
</tr>
<tr>
<td>B Implementation details</td>
<td>160</td>
</tr>
<tr>
<td>B.1 In-laboratory</td>
<td>160</td>
</tr>
<tr>
<td>B.2 IBIS case</td>
<td>162</td>
</tr>
<tr>
<td>C Test environment details</td>
<td>164</td>
</tr>
<tr>
<td>C.1 In-laboratory</td>
<td>164</td>
</tr>
<tr>
<td>C.2 IBIS case</td>
<td>165</td>
</tr>
<tr>
<td>References</td>
<td>169</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Ways of updates propagation in replicas</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Updates control in replicas</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>SRDF based data replication architecture</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Architecture for Snapshot based data replication</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>Cascade replication for PostgreSQL Slony-I</td>
<td>25</td>
</tr>
<tr>
<td>2.6</td>
<td>Database replication architecture with centralized (a) and distributed</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>(b) middleware</td>
<td></td>
</tr>
<tr>
<td>2.7</td>
<td>DARX application architecture</td>
<td>36</td>
</tr>
<tr>
<td>2.8</td>
<td>Bayou system architecture</td>
<td>42</td>
</tr>
<tr>
<td>2.9</td>
<td>Example of Regional Locking ([Source: 44])</td>
<td>45</td>
</tr>
<tr>
<td>3.1</td>
<td>General architecture of Theta replication system</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td>Communication layers for Theta replication</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>Data flow in Theta replication system</td>
<td>54</td>
</tr>
<tr>
<td>3.4</td>
<td>Middleware components</td>
<td>56</td>
</tr>
<tr>
<td>3.5</td>
<td>Transaction processing in Theta approach</td>
<td>58</td>
</tr>
<tr>
<td>3.6</td>
<td>Parameter set containing transactions information</td>
<td>61</td>
</tr>
<tr>
<td>3.7</td>
<td>Data structure containing details of transaction</td>
<td>61</td>
</tr>
<tr>
<td>3.8</td>
<td>Data structure containing details of transaction</td>
<td>62</td>
</tr>
<tr>
<td>3.9</td>
<td>TIDs selection</td>
<td>64</td>
</tr>
<tr>
<td>3.10</td>
<td>Conflict Resolution activity diagram</td>
<td>65</td>
</tr>
<tr>
<td>3.11</td>
<td>Idea of Conflict Resolution</td>
<td>67</td>
</tr>
<tr>
<td>3.12</td>
<td>Middleware transaction logging</td>
<td>68</td>
</tr>
<tr>
<td>3.13</td>
<td>Idea of transaction recovery inside middleware</td>
<td>69</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

4.1 Location of Theta method in overall replication system ........... 77
4.2 Middleware external interfaces ............................................ 78
4.3 Scalability ................................................................. 87
4.4 Centralized system versus Theta replication ............................ 90
4.5 Queries percentage influence on response times ....................... 92
4.6 Conflicts ratio influence on the approach response times ............ 94
4.7 Database size influence on mean response times – tables with indexes 97
4.8 Database size influence on mean response times – long lasting transactions for tables without indexes ............... 99

5.1 System logical structure (Source: [90]) ................................. 104
5.2 IBIS system architecture (Source: [90]) ................................ 105
5.3 Architecture of IBIS platform (Source: [65]) ......................... 106
5.4 Interactions between agents (Source: [65]) ............................. 108
5.5 IBIS databases (Source: [65]) .......................................... 110
5.6 Strategy database .......................................................... 111
5.7 Working database .......................................................... 112
5.8 Configuration database ...................................................... 113
5.9 Architecture of IBIS platform with data replication .................. 117
5.10 IBIS agents and processes communication ............................... 118
5.11 Activity diagram for IBIS replication .................................... 119
5.12 Communication in IBIS with replication ................................ 120
5.13 Deployment diagram for Theta replication management in IBIS . 122
5.14 Data flow for iBATIS data mapper (Source: [40]) ................... 123
5.15 Web Interface control panel for IBIS system .......................... 130
5.16 Crawling test architecture .................................................. 135
5.17 Data Searching test architecture ......................................... 140
5.18 Strategy management test architecture .................................. 144
5.19 Strategy management efficiency comparison .......................... 148
5.20 Listing for inserting web object – centralized database .............. 149
5.21 Listing for inserting web object – replicated database .............. 150

A.1 Elements are placed on the attached CD .............................. 156
A.2 Middleware directory structure for IBIS implementation ........... 158
A.3 Running middleware processes for replication in IBIS . . . . . . . 159
C.1 Laboratory environment for Theta replication implementation . . 164
C.2 Test environment for IBIS system with data replication . . . . . . 167
Chapter 1

Introduction

Systems used nowadays are more and more geographically dispersed. At the same
time requirement to reduce the time of access to data is getting higher and higher.

Ensuring the appropriate time of data accessibility is required in every kind
of data processing system, and it is especially important in systems implemented
for complex, distributed environments processing huge amounts of transactions
[3, 54, 61, 84]. Moreover, the growing companies expand theirs activities to many
countries all over the world and in the same time data processing requirements
and methods change as well. Additional requirements appear in such systems,
which are to ensure an appropriate level of service availability and efficiency
in environments consisting of lots of data centers spread among many distant
countries. On the other hand these demands are getting more and more difficult
to fulfil as the complexity of the system grows (multi-node environments, long
distances between remote nodes, lots of messages exchanged).

Data replication technique can be invaluable technique that allows to fulfill
high demands of distributed data management systems [3, 15, 54, 61, 84]. The
majority of the presented in the literature approaches designed for data replication
in complex, multi-node, transactional environments, focuses mainly on the fault-
tolerance issues within the system [4, 32, 38] and the security of the system [97,
116]. For instance, in multi-agent systems, which by theirs nature are distributed
systems, these approaches address the problems related to the communication
and interaction between agents, as well as the coordination of the agents [97].
It is obvious that data synchronization in distributed, multi-node systems, similarly to many other distributed systems, is performed between a large number of remote nodes. This causes that the whole system that uses the replication approach must ensure high scalability level understood as the possibility of extending of the data management system by adding remote replicas, whereas its overall performance increases or at least does not decrease [48, 54, 61].

Data in the systems covering wide areas can be stored in different database management systems running on variety of operating systems and hardware platforms, therefore, it is also necessary to enable appropriate cooperation between replication nodes in such heterogeneous systems [48].

The aim of this research is therefore to propose the replication method for distributed systems with large number of nodes that will be suitable for systems in which high level of scalability is a priority issue. The proposed method should also accomplish fault-tolerance requirements and should be applicable for heterogeneous environment based on various software and hardware platforms. The following section of this chapter introduce the research thesis and then the main activities connected with the performed research tasks are defined.

It can be revealed that Theta replication method, which enables concurrent transactions processing without necessity of usage of distributed locks, provides non-conflicting and highly scalable technique for data replication in heterogeneous database systems.

Theta replication method and related Conflict Prevention algorithm are originally proposed on the purpose of this research. The proposed method has been designed for the high availability multi-tier architecture with distributed middleware [7, 50]. In the proposed approach the users’ transactions are provided to all middleware nodes without preserving any order, where they are reassembled into the order in a way that guarantees the same overall result in all replicas. The middleware uses its own concurrency control mechanism, called Conflict Prevention algorithm, which enables transactions to be executed in parallel.

Transactions are verified against possibility of conflict appearance using Conflict Prevention algorithm, which is designed especially for Theta method pur-
poses. Conflict Prevention algorithm defines transaction processing order in a way that ensures nonconflicting processing. Moreover it supports execution of transaction with a degree of parallelism, while at the same time guarantees data consistency in replicas (the same results of processed transactions in each replica). Since Conflict Prevention algorithm ensures that conflict can not happen in any replica, which technically is a single instance database, transactions are applied to the system in a way as they would be applied in a system with centralized database. As a result usage of distributed data locks during data replication is not required.

Theta approach is suitable for heterogeneous environments which is implied as the possibility of replication implementation in environments with different RDBMS, operating systems and/or hardware platforms without the necessity of using complex gateways, additional drivers, etc.

The research contributions of this thesis and planned research tasks are:

- Analysis of replication techniques usage spectrum,
- Proposal of the new replication method – Theta replication,
- Experimental evaluation of the method,
- Practical Theta method implementation.

The following part of this section provides a short introduction to the tasks performed on the purpose of this research.

**Analysis of replication techniques usage spectrum**

In recent years many replication approaches have been proposed in the literature [17, 54, 61, 76, 77, 84], or have been implemented in database management systems used in real life solutions [42, 67, 99, 104, 105]. The review of the replication approaches, which is introduced in the chapter 2, focuses on the presentation of the relations between different types of replication. Furthermore, it allows to identify key aspects of the data replication approaches, as well as it helps to recognize and define directions for further work related to replication issues.
Comparison of conceptually similar approaches is not easy because of many subtle differences in mechanisms used and comparison of test results for various replication techniques usually is worthless since different assumptions are made for different approaches, often designed for mutually exclusive solutions (replication for OLTP systems, DSS systems, multimedia systems, etc.). It is also very difficult, and usually even not possible to perform the experiments for the available replication approaches, since the source code or deployment software of those approaches are not accessible. Therefore the review of the replication approaches presented in the chapter 2 provides the analysis of the replication approaches in the aspect of theirs usage.

**Theta replication method**

The key component of the new approach is the Conflict Prevention algorithm realized in the middleware. It determines an optimal order of transactions execution to ensure data coherence in database copies and to keep identical data in all copies. The proper order of transaction execution is determined regardless the time and the order of particular transactions appearance in the middleware. After the conflict analysis is performed, non-conflicting transactions are submitted to the database concurrently, which in fact is correct according to the 1-copy serializability theory [5], whilst conflicting transactions have to be executed apart from transactions they are in conflict with.

In the system with Theta replication implementation every location consists of an instance of the middleware software and an instance of the database management system. It is possible that both middleware and database instances work in a cluster configuration. The middleware is located between the clients submitting transactions and the database. To submit theirs transactions clients use a dedicated driver called Theta Connector. Theta Connector converts the users’ requests into the straight sets of parameters. Afterwards, these parameter sets are sent to the middleware, where conflict resolution is performed. When the conflict resolution procedure is finished, the parameters are decoded into the stored procedures calls, and then these procedures are executed in each copy of the database with a degree of parallelism.
The following list provides general assumptions and requirements for the proposed data replication approach:

- Ensuring high level of scalability,
- Reduction or elimination of the locks between remote replicas which causes that the usage of distributed transactions is not a must,
- Reduction of the amount of messages exchanged between replication nodes,
- Transaction processing in parallel,
- Replication for multiple nodes environment,
- Portability and easiness of the usage in heterogeneous environments.

Chapter 3 introduces the model and the concept of Theta replication method as well as extensive description of the approach.

**Theta method experimental evaluation**

The data management systems in the majority of the companies are required to be accessible all the time, and any unforeseen and unwanted data lose or modification are not acceptable. Thus there is no possibility to proceed research experiments in the production environment and is decided to prepare the prototype implementation of the proposed Theta replication method and conduct the appropriate experiments in the simulated environment, called in the course in-laboratory testing.

The main purpose of these experiments is verification of Theta replication method presented in chapter 3. The following factors are taken under consideration:

- High scalability level,
- Transaction processing in parallel,
- Portability and possibility of the usage in heterogeneous environments,
• Deadlock detection and resolution,
• Reduced communication,
• Resistance to failures,
• Easiness of introducing changes.

The issues related to the in-laboratory implementation of Theta replication method and its experimental evaluation are presented in the chapter 4.

Practical implementation of Theta method

System IBIS [65] is a system that continuously monitors the Internet and classifies objects found (WWW pages first of all) according to matching the content of these objects with user defined profiles which describe the area of interests. IBIS platform is a multi-agent and multi-user system. Its implementation consists of a set of machines on which processes of particular agents are run.

Performing their tasks on the purpose of the common goal, agents have to work on consistent data, easily and efficiently accessed from the servers on which processes of agents run. Because of the distributed, multi-node architecture of the IBIS system, data replication realized based on Theta replication method occurs to be ideal to fulfill these demands.

As a part of this research, data replication based on Theta method is designed and implemented for IBIS multi-agent system. Theta replication implementation for IBIS system allows to carry out evaluation of the method in real life system providing possibilities to conduct functional verification of the replication process, fault-tolerance tests and in practice scalability review.

In chapter 5 there is an introduction to IBIS platform and then there is an extensive presentation of the practical implementation of Theta replication method for IBIS system, including appropriate evaluation of the solution.

Appendices

Theta replication method is implemented and evaluated in the in-laboratory and real life environments. The wide-ranging evaluation of the implemented method
has been performed and built replication software can be used for the purposes of data sharing based on Theta replication method. It is planned to package the software and to prepare universal installer to enable easier deployment of the replication software as well as its configuration. Attached materials, described in appendices, can be used to set up replication environment as it is used within the performed evaluation of Theta method.

Appendix A provides details of the replication software implementing Theta method. Detailed specification of the software used for IBIS system implementation can be found in the appendix B, while the test environment organization including hardware specification is presented in the appendix C.
Chapter 2

Database replication

As it is stated in the introduction, majority of the currently used systems is distributed among many remote locations. Supply chain management, customer relationship management, business intelligence and many other types of applications usually do not share data or business rules nor communicate with each other to exchange data. Thus, identical data is stored in multiple locations, and as a consequence of it there is no possibility to automate business processes.

A solution to this problem can be the Enterprise Application Integration [27], which is a process of linking applications within a single organization together, to simplify and automate business processes. An access to a huge amount of data distributed among remote sites and operating on them is also realized using data replication techniques. Besides storing copies of the same data in multiple remote locations, database replication improves performance, scalability and fault-tolerance of the systems. In data replication process it is essential to keep copies consistent across the whole system. Several correctness criteria have been proposed for database replication: 1-copy serializability [5, 49], generalized snapshot isolation [1, 22] and 1-copy snapshot isolation [61].

Data replication is one of the most advantageous techniques in dealing with failures and improving efficiency for the variety of data storing systems. It has been an area of researches for almost thirty years, as the first publications related to data replication appeared in the late seventies [103, 111]. Years of detailed studies made possible a development of a wide range of algorithms and protocols which are used for maintaining data replication in distributed environments. The
Database replication definition

2.1 Database replication definition

Database replication is the creation and maintenance of multiple copies of the same database [108]. It is a technique that allows to ensure a coherence and
forces data synchronization in the distributed database system. Database copies are called replicas and may be located in many remote sites.

The database replication should be transparent for users, in a way as if they were working on a single instance database. In a system with centralized database every client connects to the same server. In a system in which database is replicated among various sites, client may choose a replica to connect, or, which is more often used in practice, client connect to dedicated replica, which usually is the nearest one to that client in term of network distance.

If replicas run on separate servers, the database replication system has two important advantages comparing to the centralized database:

- **High availability** – if one replica crashes due to a software or hardware failure, the remaining replicas can still continue processing, while centralized database system becomes completely unavailable after only one crash.

- **Increased performance** – the transaction processing load can be distributed among all the replicas in the system. This leads to a larger throughput (since queries and read operations do not change the database state, they can be independently executed in one replicas only) and a shorter response times for queries (because queries can be executed only in one replica, which is usually in the same location as the client, and without any additional communication among the replicas).

Higher availability and better performance of the system are related to the following costs:

- **Overhead related to an additional processing and communication** – the replicas require additional communication to ensure that modifications are applied to all of the database copies, which increases the load on the machines and in the communication network, thus degrades the overall system performance.

- **System complexity** – synchronization of the database copies among replicas requires usage of advanced communication and transaction processing algorithms.
2.2 Replication Model

When replicated, a simple single-node transaction may apply its updates remotely either as part of the same transaction (eager) or as separate transactions (lazy). In either case, if data is replicated at N nodes, the transaction does N times as much work [30].

Figure 2.1 shows two ways to propagate updates to replicas (eager and lazy). Eager updates are applied to all replicas of an object as part of the original transaction. One replica is updated by the originating transaction and updates to other replicas propagate asynchronously, typically as a separate transaction for each node when lazy update is performed.

![Figure 2.1: Ways of updates propagation in replicas](image)

Updates may be controlled in two ways which is presented in fig. 2.2. Either all updates are initiated by a master copy of the object, or updates may be initiated by any. Group ownership has many more chances for conflicting updates.

The considered model assumes that the database consists of a fixed set of objects. There are a fixed number of nodes, each storing a copy of all replicated objects. Each node originates a fixed number of transactions per second. Each
transaction updates a fixed number of objects. Inserts and deletes are modeled as updates, reads are ignored.

Replica update requests have a transmit delay and also require processing by the sender and receiver. These delays and extra processing are ignored; only the work of sequentially updating the replicas at each node is modeled. Some nodes are mobile and disconnected most of the time. When first connected, a mobile node sends and receives deferred replica updates. Parameters used in replication model are listed in table 2.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Size</td>
<td>Number of distinct objects in the database</td>
</tr>
<tr>
<td>Nodes</td>
<td>Number of nodes; each node replicates all objects</td>
</tr>
<tr>
<td>Transactions</td>
<td>Number of concurrent transactions at a node. This is a derived value</td>
</tr>
<tr>
<td>TPS</td>
<td>Number of transactions per second originating at this node</td>
</tr>
<tr>
<td>Actions</td>
<td>Number of updates in a transaction</td>
</tr>
<tr>
<td>Action_Time</td>
<td>Time to perform an action</td>
</tr>
</tbody>
</table>

Table 2.1: Variables used in the model and analysis

Each node generates TPS transactions per second. Each transaction involves a fixed number of actions and each action requires a fixed time to execute. Thus
duration of transaction equals to $Actions \cdot Action\_Time$. Given these two observations, the number of concurrent transactions originating at a node is:

$$Transactions = TPS \cdot Actions \cdot Action\_Time \quad (2.1)$$

In a system of $N$ nodes, $N$ times as many transactions will be originating per second. Since each update transaction must replicate its updates to the other $(N - 1)$ nodes, it is easy to see that the transaction size for eager systems grows by a factor of $N$ and the node update rate grows by $N^2$. In lazy systems, each user update transaction generates $N - 1$ lazy replica updates, so there are $N$ times as many concurrent transactions, and the node update rate is $N^2$ higher. This non-linear growth in node update rates leads to unstable behavior as the system is scaled up [30].

Eager Replication updates all replicas when a transaction updates any instance of the object. There are no serialization anomalies (inconsistencies) and no need for reconciliation in eager systems. Locking detects potential anomalies and converts them to waits or deadlocks [30].

In a single-node system the with eager replication transactions have about $\frac{Transactions \cdot Actions}{2}$ resources locked (each is about half way complete).

Since objects are chosen uniformly from the database, the chance that a request by one transaction will request a resource locked by any other transaction is $\frac{Transactions \cdot Actions}{2 \cdot DB\_Size}$. A transaction makes $Actions$ such requests, so the chance that it will wait sometime in its lifetime is approximately [29, 31]:

$$PW = 1 - (1 - \frac{Transactions \cdot Actions}{2 \cdot DB\_Size}) \cdot Actions \quad (2.2)$$

$$PW = \frac{Transactions \cdot Actions^2}{2 \cdot DB\_Size} \quad (2.3)$$

A deadlock consists of a cycle of transactions waiting for one another. The probability a transaction forms a cycle of length two is $PW^2$ divided by the number of transactions. Cycles of length $j$ are proportional to $PW^j$ and so are
2.3 Development of replication techniques

Even less likely if $PW << 1$. The probability that the transaction deadlocks is approximately:

$$PD = \frac{PW^2}{\text{Transactions} \times \text{Actions}^4} \quad (2.4)$$

$$PD = \frac{\text{TPS} \times \text{Action\_Time} \times \text{Actions}^5}{4 \times \text{DB\_Size}^2} \quad (2.5)$$

Lazy group replication allows any node to update any local data. When the transaction commits, a transaction is sent to every other node to apply the transaction updates to the replicas at the destination node. It is possible for two nodes to update the same object and race each other to install their updates at other nodes. The replication mechanism must detect this and reconcile the two transactions so that their updates are not lost.

Transactions that would wait in an eager replication system face reconciliation in a lazy-group replication system. Waits are much more frequent than deadlocks because it takes two waits to make a deadlock [30]. Eager replication waits cause delays while deadlocks create application faults. With lazy replication, the much more frequent waits are what determines the reconciliation frequency. So, the system-wide lazy-group reconciliation rate follows the transaction wait rate equation:

$$\text{Lazy\_Group\_Reconcil\_Rate} = \frac{\text{TPS}^2 \times \text{Action\_Time} \times (\text{Actions} \times \text{Nodes})^3}{2 \times \text{DB\_Size}} \quad (2.6)$$

2.3 Development of replication techniques

At the beginning of its development data replication for databases was realized by the modifications in a source code of the database management system engine. These changes were performed in various parts of the engine, like in the transactional log module which consists of all the modifications in database, or with the usage of additional modules ensuring group communication. An example of the system based on this idea is the implementation of the system Postgres-R [53, 54], which is characterized by relatively good performance as the overhead
2.3 Development of replication techniques

related to replication process is low. However, since it is necessary to modify the source code of the essential part of database, which is its engine, systems implemented on the basis of code modifications in the database engine are very hard to realize in different system platforms. Difficulties are present not only in case of the usage of different system engines or operating systems, but also appear for different versions of the database supplied by the same vendor, even if it is the next release of this database.

The solution presented in [5] is based on the propagation of all local operations to each remote site in the system. Unfortunately, propagating of every operation to the remaining sites in the system led to the frequent occurrences of the distributed deadlock for these operations. Thus, after some research a new ROWAA approach (Read One Write All Available) ensuring the coherency of replication process was designed. In this approach all operations related to a single transaction are first processed in one site only, and afterward data modifications are transferred to the remaining nodes, without the necessity of any additional communication messages. Since fewer messages are transferred through the network to process a transaction, it is obvious that the transaction time is shorter, and the replication performance is improved. Optimistic 2 Phase-Locking protocol (O2PL) presented in [9] is the example of this approach. It is one of the first approaches ensuring replicated data coherence on the basis of ROWAA approach. This approach is realized with the usage of an adaptation of the two phase-locking protocol (2PL) in which local transactions are distinguished from remote transactions. It allows to predict and avoid or decrease the quantity of deadlocks. Start of the transaction processing immediately after it is delivered to the site allows on significant decrease of the overall time required to commit this transaction. After a complete transaction is delivered, its correct order is determined, and if the transaction is not in conflict with preceding transactions it is committed. Otherwise, the whole transaction is processed from the beginning once more.

To decrease deadlocks influence on data replication approaches based on group communication were designed and implemented. Group Communication Systems (GCS) approaches used in [11, 43] provide a mechanism that guarantee necessary order of delivered messages in the network, and also enable failure detection in any site of the system. The most restrictive order of delivered messages requires
the same order of delivering in every node of the system, which allows to avoid distributed deadlocks. Group communication based approaches were widely explored and also some systems using it were implemented, for instance Basic Replication Protocol (BRP) presented in [43] or Postgres-R project in [53, 54].

Unfortunately, group communication based approaches are not the most effective technique in preventing deadlocks for all possible applications or every type of transaction. It is so, because exchanging of additional messages between sites is necessary to ensure the proper order of delivered messages related to transactions. An overhead related to it can significantly decrease the performance of the approach [18]. The following researches in GCS techniques led to reduction of the influence of latencies in the network on the overall time required to deliver transaction to every node in the correct order. An example of such solution is Generic Broadcast approach discussed in [2, 78]. In this approach delivery order is important only for these transactions which are conflicting, while the rest of transactions can be delivered in any order. Another example is Optimistic Atomic Broadcast approach presented in [55, 115]. Messages in this approach are delivered in the same order as they were received, which enables quick application of writeset in the remote nodes, despite of the necessity of waiting for the final order of transactions before they are committed. Thus, only those remote transactions whose writeset did not follow the total order are rolled back, reapplying them in the correct order. Group communication based techniques also have been used for the epidemic algorithms presented in [37].

2.4 Review of the approaches

Very well known and useful classification was proposed by Gray in [30], where replication approaches are grouped according to two parameters. These parameters are: the place in which a transaction is initiated and the time when a transaction is distributed to each node. Wiesmann in [117, 118] proposes an extended classification using three parameters – first parameter is the architecture of the server, the second one is the degree of communication among database nodes during the execution of a transaction, and the last one is the transaction termination protocol.
2.4 Review of the approaches

The classification of replication techniques is included in the following subsections. Replication techniques are distinguished on the basis of an architecture of the replication system, a time of data propagation between replicas, interactions between nodes, a way of termination of transactions and a possibility of its usage in heterogeneous environments.

2.4.1 Time of transactions

In replicated system consistency of data in all replicas, in the presence of updates, has to be ensured. Some replication protocols require strong consistency, which means that data must be consistent all the time. In such cases replicas must coordinate updates before the response is sent to the client and response time increases. There are protocols that require only weak consistency allowing data to be inconsistent temporarily. This let us to get faster response for writes, but it is possible that transactions read data which is not up to date since it have not yet been applied in every remote replica. Weak consistency protocols may even require to rollback updates previously applied and committed, even though the client received a confirmation that commit was successful.

Depending on the time when modification in one site is propagated to other replication sites two replication approaches are distinguished: eager and lazy. In eager approach transaction may be committed only when it is possible to commit it in all sites. Unlike eager approach, lazy approach allows updates to be committed before they are propagated to other sites.

2.4.1.1 Eager replication

Eager replication (synchronous) approach requires immediate propagation of changes from the database node where transaction was submitted to all nodes in the replication system. Thus, eager replication keeps all database replicas totally synchronized in all nodes by updating all the replicas as a part of one atomic transaction. Eager replication ensures serializable execution of transactions which causes that there are no concurrency anomalies. However, eager replication reduces update performance and increases transaction response times because extra updates and messages are added to the transaction [30].
2.4 Review of the approaches

Eager replication implementations most often are based on Two-Phase Commit protocol (2PC) [5] or some modifications of this protocol – Three-Phase Commit protocol (3PC) [98], etc. Two-Phase Commit protocol is a distributed algorithm that need all nodes in a distributed system to agree to commit a transaction. The protocol results in either all nodes committing the transaction or aborting otherwise, even in the case of network failures or node failures.

The greatest drawback of the two-phase commit protocol is the fact that it is a blocking protocol as it usually uses Two Phase Locking protocol (2PL). In 2PL protocol a node blocks data (raws, table) while it waits for a message, which means that other processes competing for resource locks held by the blocked processes will have to wait for the locks to be released.

Ensuring 1-copy serializability, which guarantees that data is coherent and integral, is the most important advantage of eager replication. Data is up to date in each node and all the time. Disadvantages of this technique are reduced scalability and low efficiency. The consequence of locks is the possibility of deadlocks, which also significantly decrease performance.

Eager replication protocols ensure strong consistency and fault-tolerance as updates must be confirmed at remote replicas before replying to clients. However, this have a meaningful impact in user visible performance. Also in contrast with lazy protocols, approaches to eager replication provide very different trade offs between performance and flexibility [46].

The following part of this subsection provides popular commercial implementation of eager replication approach.

**Volume Replication**

Replication of disk volumes performed at the block I/O level is a straightforward replication approach for general purposes. By intercepting each block written by the application designated volumes and shipping it over to network, a remote copy is maintained ready for fail-over. Reads are performed on the local copy. The replication process is thus completely transparent to the application.

The downside of the approach is that remote updates are performed with a block granularity, which depending on the application might represent a large
2.4 Review of the approaches

network overhead. The approach is also restricted to fail-over, as the backup copy cannot usually be used even for read-only access, due to lack of cache synchronization at the application level.

Examples of volume replication solutions can be found in Veritas Volume Replicator [106] and EMC Symmetrix Remote Data Facility [23] available for many different operating systems, as well as in the open source DRBD for Linux OS [33].

RAIDb protocols

A Redundant Array of Inexpensive Databases (RAIDb) provides a better performance and fault tolerance than a single instance database. It is achieved at a lower cost by combining multiple database instances into an array of databases. Like RAID to disks, different RAIDb levels provide various cost/performance/fault-tolerance tradeoffs. RAIDb-0 features full partitioning, RAIDb-1 offers full replication and RAIDb-2 introduces an intermediate solution called partial replication, in which the user can define the degree of replication of each database table [46].

Figure 2.3: SRDF based data replication architecture
In RAIDb the distribution complexity is hidden from the clients and therefore they are provided with the view of a single database like in a centralized architecture. As for RAID, a controller is located in front of the underlying resources and the clients send their requests directly to the RAIDb controller, which distributes them among the set of RDBMS. RAIDb is implemented as a software solution in C-JDBC [107].

2.4.1.2 Lazy replication

In lazy replication (asynchronous) the updates of a transaction are propagated once it has already committed. Lazy approach does not require continuous connection between all nodes. Every node works independently and reads and writes are processed locally. Updates are propagated to remote nodes either as SQL statements or as log records containing the results of executed operations.

Despite many disadvantages, lazy approach has been used in several commercial DBMSs and it is the important option when mobile or frequently disconnecting databases are considered. Lazy replication is used to synchronize data for certain tables only, not for the whole database instance. Synchronization can be executed periodically, on the user’s demand or just after successful modification in local node.

Processing transactions locally allows on quickly transaction completion, but does not always ensure replicas consistency and may lead to conflicts, data incoherence and high abort rate. Advantages of lazy replication approach are: scalability, no overhead related to Two Phase Locking, resistance to single node failures, constant connection among all nodes not required.

Lazy replication has become a standard feature of many commercial database management systems and open-source database projects [42, 63, 67, 99, 105]. In the replication system with lazy approach implementation every transaction is executed and committed at a single replica without synchronization with other replicas. Every other replica is synchronized later as changes are captured, distributed and applied. Because of it, the overall performance of the whole system measured as a time of transaction processing increases significantly. However,
since updates can be lost after the failure of any replica, lazy replication approaches are not suitable for fault-tolerance solutions with strong data consistency required [46].

There is a variety of implementations in which lazy replication operates in different ways. Main differences between implementations of lazy approach exist in the following areas:

- the way of the whole system is managed,
- system architecture determining which replica processes and publishes updates to other replicas,
- the way in which updates are captured, distributed and applied,
- filtering of the processed transactions.

Lazy replication protocols can be divided in three phases:

- Capture – updates performed on replicated objects transformed into a format suitable for publication,
- Distribution – changes in published objects are propagated to the relevant replicas,
- Apply – updates applied in the relevant replicas.

The following part of this subsection contains a general description of the most popular commercial database management systems with implementation of lazy replication approach.

**Oracle**

Oracle database in version 11g and 10g offers three basic replication solutions [8, 67, 68]:

- Snapshot replication,
- Streams replication,
2.4 Review of the approaches

- Advanced replication.

**Snapshot replication** also called materialized view replication is based on the capturing of changes for the materialized views, which are then propagated and applied in the remote sites. A snapshot is a query that has its data materialized, or populated in a form of a data table. When a snapshot is created a table corresponding to the column list in the query is created. When the snapshot is refreshed, that underlying table is populated with the results of the query. For replication, as data changes to a table in the master database, the snapshot refreshes as scheduled, and that data are distributed to the replicated databases.

![Architecture for Snapshot based data replication](image)

Figure 2.4: Architecture for Snapshot based data replication

Oracle **Streams Replication** enables the propagation and management of data, transactions and events in a data stream either within a database, or from one database to another. Streams replication passes published updates, events or even messages to subscribed destinations, where they are applied or further processed.

The Oracle Streams technology uses the operation logs (redo logs) as an input for a capture process. The capture process formats both operations on data
(DML) and data definitions (DDL) into the events which are enqueued for distribution process. The distribution process reads the input queue and enqueues the events in a remote database. The queue from which the events are propagated is called the source queue, while the queue receiving the events is called the destination queue. There can be a one-to-many, many-to-one, or many-to-many relationship between source and destination queues. Depending on the type of events an apply process at the destination site applies them directly to the database, or may dequeue these events and send them to an apply handler which performs customized processing of the event and then applies it to the replicated object.

Oracle Advanced Replication is realized as a set of support triggers and procedures for each replicated object. These triggers and procedures enqueues a specific remote procedure calls according to the command executed. This queue is consumed by the Oracle implementation of the distribution process which push and pull the deferred calls, propagating the changes from sources to destination databases. Then, the apply process dequeues these information and updates the subscriber.

Since Oracle Advanced Replication allows to perform updates in every replica, a conflict detection mechanism is used in every site. When a conflict is detected, actions defined for the specific types of conflict are performed (conflict resolution procedure).

**MS SQL Server**

MS SQL Server 2005/2008 provides three replication solutions: transactional replication, snapshot replication and merge replication [63, 101, 104].

**Transactional Replication** uses the log as a source to capture incremental changes that were made to a published objects. The capture process copies transactions marked for replication from the log to the distribution agent. Basically, the capture process reads the transaction log and queues the committed transactions marked for replication. Then, the distribution agent reads the queued transactions and applies them to the subscribers.
In **Snapshot Replication** the entire published object is captured thus it distributes data exactly as it appears at the specific moment in time. This solution does not monitor the updates made against an object. Roughly, the capture process is implemented by a snapshot agent. Periodically, it copies the replicated object schema and data from the publisher to a snapshot folder for future use by a distribution agent, which also acts as an apply process.

Snapshot replication can be used by itself but the snapshot process, which creates a copy of all of the objects and data specified by a publication, is very often used to provide the initial state of data and database objects for the transactional and merge replication.

**Merge Replication** allows the publisher and the subscribers to make updates while they are connected or disconnected. When both are connected, it merges the updates. The capture process tracks the changes at the publisher and at the subscribers, while the distribution process distribute changes and also acts as an apply process.

Merge replication allows various sites to work autonomously and later merge updates into a single, uniform result. Because updates are made at more than one node, the same data may have been updated by the publisher and by more than one subscriber. Therefore, conflicts can occur when updates are merged and merge replication provides a number of ways to handle conflicts.

**IBM DB2**

IBM DB2 Universal Database 8.2 provides two different solutions that can be used to replicate data from and to relational databases: SQL replication and Q replication [41, 42].

In **SQL Replication** changes are captured at sources and staging tables are used to store committed transactional data. The changes are then read from the staging tables and replicated to corresponding target tables. With staging tables, data can be captured and staged once for delivery to multiple targets, in different formats, and at different delivery intervals.

**Q Replication** is the replication solution that can replicate large volumes of data at a very low levels of latency. Q replication captures changes to the source
2.4 Review of the approaches

tables and converts committed transactional data to messages. This data is sent as soon as it is committed at the source and read by Q replication. The data is not staged in tables. The messages are sent to the target location through the message queues. These messages are then read from the queues at destination sites, converted back into the transactional data and applied to the target tables.

IBM DB2 UDB version 8.2 also provides a solution called **event publishing** for converting committed source changes into messages in an XML format and publishing those messages to applications such as message brokers.

**PostgreSQL**

**Slony-I** is a replication solution for the PostgreSQL database, which implements lazy replication as a "master to multiple slaves" replication [99].

In Slony-I, the capture process is implemented using triggers which log the changes made against the published objects. These changes are then periodically distributed using a replication daemon which connects directly to the publisher, reads the logged changes and forwards them to the subscribers.

![Figure 2.5: Cascade replication for PostgreSQL Slony-I](image)

Slony-I allows to connect several subscribers in cascade. The management process is not integrated or centralized, thus the maintenance tasks must be done in each replica separately.
2.4 Review of the approaches

MySQL

MySQL includes a built-in lazy replication protocol that can be configured to propagate updates between replicas [64]. Replication enables data from one MySQL database server (called the master) to be replicated to one or more MySQL database servers (slaves). Replication is asynchronous and replication slaves do not need to be connected permanently to receive updates from the master.

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves.

2.4.2 System architecture

The first parameter to consider in database replication classification is the place where transactions start to execute. Depending on transactions location in primary approach updates can be executed only in primary (master) site, whilst update everywhere approach allows on update execution at any site (usually updates at the user’s local site). According to propagation time updates done at any site have to be propagated to other sites.

Primary copy approaches are designed for Master-Slave architecture and require to have a specific node (the primary copy) associated with all data in system. Each data update, before being sent to all nodes, has to be sent to the primary node, where it is processed – executed or analyzed to determine the order of execution. After transactions processing the primary copy propagates update or its results to the remaining nodes.

Replication mechanism in primary copy approach does not require distributed protocols like Two-Phase Commit (2PC) or Two Phase Locking (2PL), which eliminates overhead related to these protocols. There are also eliminated conflicts in transactions in primary copy approach as transactions are processed only in primary node and conflict resolution is realized in a single instance database. On the other hand, this technique has many drawbacks which prevent it from being used widely. These disadvantages are: single point of failure connected with one
primary node, bottlenecks in primary node and read-only access to data in nodes that are not primary copy.

Since primary copy approach enforces a bottleneck in the replication system and a single point of failure, some modifications for the approach are used to overcome these limitations. This approach easily overloads the primary node, therefore read-only transactions can be applied in secondary nodes which balances the load. Bottlenecks can also be avoided by data partitioning among more than one node. Having different primary nodes for subsets of data, update transactions can be executed in parallel, dividing the load among these nodes. Furthermore, in case of crash of the primary copy, one of the other nodes can become primary one.

Therefore, primary copy approach is mainly used for fault tolerance. In case of one node crash the database is still available for users. This technique is applicable for data transfer between Online Transactional Processing systems (OLTP) and Decision Support Systems (DSS).

Update everywhere approach is related to Multiple Master architecture. This database replication technique does not impose limitations on a node where updating transactions can be processed, and updates can be performed in each node in the system. In update everywhere approach updates can reach two different nodes at the same time, even though they are conflicting, which is opposite to primary copy approach. In update everywhere replication nodes are equal to each other. Each of the client’s transaction is processed by one node and then changes are propagated to other replicas. However, updating data in every node causes appearance of conflicts between transactions. Conflict resolution approaches are used to deal with conflicts.

Update everywhere approaches are suitable for dealing with failures, as election protocols are not necessary to continue processing. Another update everywhere approach advantage related to the symmetric architecture of this replication system is the possibility of reading and writing data in every replica.

When update everywhere technique is used in connection with eager approach it usually require usage of Two-Phase Commit or Two Phase Locking protocols, which significantly decreases overall performance of the system. Replication conflicts appearing when approach is used in connection with lazy replication are also
serious problem, as they lower replication performance and need special treatment. The complexity of concurrency control in update everywhere approach requires additional maintenance, and usually is realized in middleware subsystem, which coordinates global transactions.

### 2.4.3 Middleware based replication

It is possible to find many algorithms and protocols which are used for maintaining data replication in distributed environments [3, 43, 61, 76]. To ensure data consistency, concurrency control in database management systems is based either on internal mechanisms of database management system, or uses a middleware tier [7, 50]. Concurrency control based on middleware protocols is getting more and more popular because of its flexibility and simplicity.

Unlike systems with modified code of the database engine, replications system built in middleware architecture make possible realization of replication that is transparent for the users and applications. If middleware based replication is implemented it is not necessary to do any modifications in the code of the database engine.

![Database replication architecture with centralized (a) and distributed (b) middleware](image)

Figure 2.6: Database replication architecture with centralized (a) and distributed (b) middleware

In recent years appeared many middleware based replication protocols for databases. The middleware protocol maintain replication control in middleware
2.4 Review of the approaches

subsystem placed between client and databases. Middleware based replication can be developed and maintained irrespective of database management systems, and there is a possibility to use it in heterogeneous environments.

Distributed middleware architecture has many advantages in regard to replication control, which causes that this architecture has been chosen for further research. The most important features of the middleware based replication are:

- Increased fault tolerance of the system – in case of failure of one of the middleware component it is easily replaced.

- Scalability as new databases and middleware servers can be easy added in a location it is required.

- Possibility of online upgrades and maintenance.

- Increased performance of the middleware.

2.4.4 Server interaction

The following section provide more information on the classification based on server interaction, parameter related to the number of messages which are sent among database nodes to ensure the applying of transactions. This interaction impacts amount of network traffic, and in consequence is a significant overhead to the processing of transactions. Since applications are more and more geographically dispersed and distances among nodes are getting extremely large, network communication becomes an important factor in replication process.

Server interaction is explained as a function of the number of messages necessary to handle the operations of a transaction [118]. The number of network interactions influences replication protocol significantly and determines the order in which transactions are to be processed. On the basis of interaction among nodes two cases are distinguished: constant interaction and linear interactions.

In constant interaction the number of messages that are used to synchronize database nodes is constant and does not depend on the number of particular operations in the transaction. Typically, these approaches merge all operations related to the transaction in a single message and use only one message per
transaction. Replication techniques might also use a greater than one message, but for constant interaction approach number of messages is always unchangeable, independently of the complexity of the transaction.

On the contrary, linear interaction enables technique in which processing of transaction requires exchanging variable amount of messages. The amount of messages depends on the number of operation in particular transaction as it is usually proportional to the amount of single operations which are parts of the entire transaction. These single operations might be simply SQL statements, changes collected in the writesets or records in database logs which contain the results of transaction executed in specific node.

### 2.4.5 Transaction termination

The way of transaction termination decides how atomicity is guaranteed. It leads to the existence of two replication techniques. One of them needs exchange of messages to fulfill requirements of the ACID, the other allow to satisfy ACID properties without any additional transmission of messages [117].

Approaches based on voting termination requires an additional messages to terminate transaction for establishing a coherent state of the replicas. In voting protocols the decision to abort transaction can be made by primary node (weak voting) or by any replica (strong voting).

Voting messages might be very straightforward as a single message sent by one replica to confirm its state. On the basis of these message the other replicas determine their states. Message exchange can also be much more complex as in case of protocols like two-phase commitment protocol.

Non-voting termination techniques allow replicas in all nodes to decide on their own whether they commit the transaction or abort it. However, transactions have to be executed in the same order in each replica. It is only a semblances that this is extremely strict requirement to ensure the same order of transactions processing, because for non-conflicting transactions identical order is not necessary.
2.4.6 Environment complexity

The architectures of the replication environment might be quite simple as in systems with all replicas configured in exactly the same way. These architectures can also be extremely complex in connection with many layers used in such systems, like client, middleware and database tiers. Requirement of cooperation and realization of replication among different database engines used in one replication system is a great challenge. Depending on an approach possibility to be adapted for replicating data among miscellaneous types of databases, we can distinguish two approaches: heterogeneous and non-heterogeneous replication [48].

Non-heterogenous approaches are the approaches working in environments consisted of each replica implemented upon the same type of database. This kind of replication is much easier to implement than replication for heterogeneous databases. However, the requirement of replication among different types of databases is getting stronger and stronger and heterogeneous replication is the area for future research.

Heterogeneous approaches can be used in systems with a number of varied databases, which means all these database might be supplied by different software vendors. Especially, it might happen that these databases have different APIs. Thus, realization of the replication for heterogeneous databases is very complex and demanding task.

There are some interfaces for accessing data in a heterogeneous environment, like Open DataBase Connectivity driver (ODBC) and Java DataBase Connectivity (JDBC). They can be used for accessing data among replicas, however, it adds another layer, in already complicated environment, and might adversely affect replication performance. Object-relational mapping libraries represented by Enterprise Objects Framework, Hibernate, Enterprise Java Beans and so forth, which convert data between incompatible databases and object-oriented programming languages are also possible to use for replication, but overhead connected with objects mapping might be too high reducing replication effectiveness in practice.
2.5 Review of replication techniques usage

Database management systems used in present day systems can be classified according to the way of data management as operational and analytical databases. Operational databases are used to perform everyday duties in many organizations, institutions and companies. They are applied not only in the systems which require to gather and store data, but in the systems which need to modify data as well. Operational database stores dynamically changed data which reflects actual state of reality.

On the other hand, analytical databases are used to store historical and archival data, or information related to some events. When a company wants to analyze market tendencies, to gain access to long-term statistic data, or to perform business forecasts, then uses data stored in analytical databases. Data maintained by analytical databases are hardly ever modified, if modified at all. Moreover, this data always presents state of objects in some established moment in the past.

Furthermore, appearing new technologies and data formats related to designed database management systems, more and more frequently require storing diverse data types like sounds, graphics, animations and videos, while storing traditional text and numbers too. Existing applications are continuously developed using modern, advanced relations between different types of data, which makes it easier to search demanded data or objects.

The following subsections provide review of the database management systems for different types of application in the aspect of the usage of data replication in those systems. Replication aspects introduced in the review are related to transactional and reporting data processing as well as storing large objects, mobile data, real time data or spatial data.

2.5.1 Transactional data processing

The characteristic feature of On-line Transaction Processing systems (OLTP) is that they are used to process large volumes of simple, read or write transactions. Transactional systems, which operate on huge amount of data, are nowadays
applied in many implementations of banking systems, telecommunication, reservation systems (flight tickets for instance), etc.

The emphasis in OLTP systems is put on retaining data integrity in multi-users environments, as well as on an efficiency of data processing measured as a number of transactions per time unit. In a typical OLTP system database efficiency is a key factor since users demand fast answers and reactions during performed operations, despite the fact that large number of transactions causes downgrades in performance related to the appearance of disks contentions, resources locking, deadlocks, etc. Quality and usability of such systems is highly dependent on the rate of database writes (writes on disks or other I/O devices). To fulfill excessive requirements for databases in OLTP systems, they are usually designed to store as little data as possible and the time for INSERT, UPDATE or DELETE operations is minimized. Additionally, data replication can be applied to improve performance and increase availability of the transactional system.

When data replication is used in OLTP systems, then it is necessary to continuously ensure data consistency between all copies of data. In literature there are many replication approaches that could be used for data replication in OLTP systems. Such approaches can be found in [3, 15, 54, 61, 84]. An approach that is nowadays very often used for commercial OLTP systems is the eager replication approach based on 2-Phase Commit Protocol (2PC) described in [5]. These approaches ensure data consistency and fulfill demands of 1-copy serializability [49]. However, since the necessity of the locking of resources distributed among many remote locations, response times and the risk of occurrence of distributed deadlocks increases, which leads to the lower scalability of such systems. Thus, such systems are usually restricted to consist of few nodes only, which operate not very far from each other.

In general, replication for OLTP systems is implemented using an architecture with equivalent nodes (multi-master, update everywhere replication). This is caused by requirement of data consistency in every replica. OLTP systems with equivalent nodes allow to read and write data in any replica in the whole system. Since data in each copy is identical, they are also suitable to ensure high resistance to failures (fault-tolerance). In OLTP systems with replication based on 2PC, their efficiency is usually significantly reduced by distributed locking
and deadlocks. Another drawback of the eager replication with equivalent nodes is the necessity of conflict resolution, which causes the implementation of such system more difficult and impairs an efficiency of such solutions.

In systems built in architecture in which one node is more important than others (primary copy, master-slave replication), data at first is modified in the main node and only after the transaction completion it is distributed to the other nodes. This leads to data inconsistency between particular replicas and as a consequence master-slave architecture is not suitable for OLTP systems. Lazy replication approaches are also not feasible for replication in OLTP systems. This is a result of a deferred propagation of changes which does not ensure consistent data state in each copy of data. It causes that conflicts appear, which very often leads to the necessity of performing complex conflict resolution procedures [16, 20, 95, 110]. It is obvious that all that factors cause a significant decrease in an overall performance of such implementations.

Data processing in transactional distributed multi-node systems

One of the most important feature of any distributed system is its transparency which is seen by a user as if it was single, integrated system. A distributed system may have a common goal, such as solving a large computational problem. Alternatively, each computer may have its own user with individual needs, and the purpose of the distributed system is to coordinate the use of shared resources or provide communication services to the users [28, 79]. Large volume of transactions and huge amount of data stored in such systems require complex solutions to achieve appropriate performance, data availability and security.

Distributed cooperative applications (e.g., e-commerce, air traffic control, city traffic management) are now designed as a sets of autonomous entities called agents [21]. A system with number of interacting agents is named a multi-agent systems. Distributed multi-agent systems has became a powerful tool in the process of designing scalable software. The general outline of a distributed agent software consists of computational entities which interact with one another towards a common goal that is beyond their individual capabilities [32]. In the
2.5 Review of replication techniques usage

following considerations multi-agent system is used as a representation of distributed system with multiple nodes (agents) in remote and very often distant locations.

Many different definitions of the notion of software agent can be found in literature [25, 32, 58]. The main properties of the multi-agent systems are:

- defined individual goals and resources,
- ability to do specified work consisted of particular tasks realized by certain agents,
- capability to act in the near environment which includes communication with other agents,
- ability to perform actions with some level of autonomy and/or reactivity,
- capacity to provide services.

These features cause that the agent model is also very suitable for building adaptive applications, where interactions between the different entities involved may be altered during the computation, and where this change must have an impact on the software behavior. In applications built on the basis of the multi-agent model, roles and relative importance of the agents can greatly vary during the agents interaction and cooperation as well as the course of computation. Therefore, the agents need to be able to change roles, plans and strategies. In addition, new agents may join or leave the application. It causes that it is very difficult, or in many cases it is even impossible to identify in advance the most critical software components of the application. It is even getting more important in large scale systems, where environment characteristics can vary a lot from one network area to another.

Consistency between replicas in distributed, multi-node systems can be maintained using the following two strategies: the active one in which all replicas process all input messages concurrently, and the passive one in which only one replica processes all of the input messages and periodically transmits its current state to the other replicas. Active replication is dedicated to those applications which require full recovery over short delays. Passive replication makes for lower
overhead in a failure-free environment but recovery is less efficient. The choice of the most suitable strategy is directly dependent on the environment context, especially the failure rate, the kind of failure that must be tolerated, and the application requirements in terms of recovery delay and overhead.

As the amount of locations and agents within the multi-agent system increases, so does the probability of failures, and so does the requirement for system scalability. It is therefore crucial to apply reliable and scalable protocols when implementing agent based solutions for large scale systems. One way to achieve such results is replication of agents which is an efficient way to achieve fault tolerance in scalable agent systems. Presented in fig. 2.7 DARX architecture for fault-tolerant agent computing is defined as a software component that possesses a representation on two or more hosts [32]. DARX allows to replicate software elements on the spot and it is possible to change the current replication strategies on the fly.

![Figure 2.7: DARX application architecture](image)

DARX architecture authors emphasize the fact, that replicating every agent over a large-scale network would lead to excessive overheads, both in terms of network load and computation time. Thus, the proposed approach is suitable for systems in which at any given point of the computation, only a small subset of agents is likely to be crucial to its continuation. Moreover, only the specific
agents, which are temporarily identified as crucial to the application, should be replicated and the applied strategy should be carefully selected.

Majority of the approaches presented in literature focus mainly on the the fault tolerance problem within a multi-agent system [4, 32, 38]. These approaches address the problems related to communication and interaction between agents, as well as coordination of the agents. The other category of the multi-agent approaches addresses the difficulties of making reliable mobile agents which are more exposed to security problems [97, 116].

2.5.2 Large amount of data

Decision Support System (DSS) is the system that supplies information and knowledge to support analysts or management in a process of decision making. Nowadays DSS systems are used in a number of companies in every field of economy. Based on the analysis of data stored in data marts, companies can define their strategies and actions to increase sales, decrease costs, improve clients service, organize marketing campaigns and many others.

Reporting and analytical system implementations are very often based on structures called cubes which were defined for Online Analytical Processing (OLAP) [13]. Cube is such a data structure that allows fast analysis of data. It can also be defined as the capability of manipulating and analyzing data from multiple perspectives. The arrangement of data into cubes allows to overcome a limitation of relational databases. OLAP cubes can be thought of as extensions to the two-dimensional array of a spreadsheet. For example a company might wish to analyze some financial data by product, by time-period, by city, by type of revenue and cost, and by comparing actual data with a budget. These additional methods of analyzing the data are known as dimensions.

In opposite to OLTP system, DSS is characterized by relatively small amount of transactions, most of which are complex read transactions. Efficiency measure for DSS system is its response time. The most often DSS systems are built as data marts, where data is organized in multidimensional structures, in which facts (numbers stored in database) depend on a number of parameters called dimensions.
2.5 Review of replication techniques usage

Decision support systems are usually used as a repository of historical data. This historical data stored in these systems is not modified after it is loaded, while new data is appended to the system in a form of data loading. Loaded data is previously extracted from a source system, for instance from OLTP system, and then is supplied as flat files, throughout additional gateways, and so on. Also data replication approaches can be used to load data to DSS systems. In such configuration data is replicated directly to the data marts tables. Architecture of the data replication for DSS is very often based on master-slave architecture in which a source OLTP system (or more OLTP systems) is a master node, and DSS system (data destination) is a slave node. Since data copies stored in DSS systems can be refreshed with some delay comparing to the source systems, in the majority of cases lazy (asynchronous) replication is the best choice for DSS systems. DSS systems require only a part of data stored in source systems, therefore techniques based on partitioning [66] and partial replication [15] can also be used to achieve desired results.

In most cases problems arise while data from different sources is to integrate data under a given target schema. The problem is getting more complicated when the complexity of data grows as for instance in systems storing spatial or XML objects. Additional information on solutions and data integration for spatial data can be found in [87] and for XML data in [73].

To refresh data in commercial DSS systems hardware replication is very often used. Before data loads are initiated, synchronization between source and destination I/O devices is suspended, and data stored in a slave location are used for loading DSS. There are many commercial implementations that allow to perform copies on hardware level or backup copies (standby) which can be used for data copying to DSS systems, only slightly absorbing source systems (in some cases it is even possible that this process has no influence on the source system). Commercial database vendors offer variety of solutions that can be used for refreshing data in DSS systems [42, 63, 67, 99, 105].

Many approaches that can be used for data replication to DSS systems, can also be found in literature [15, 17, 59, 60, 72, 76, 89, 117].
2.5.3 Multimedia data

The majority of currently used replication techniques requires a usage of lock mechanisms and it is necessary to transfer all data related to the processed transactions. However, such approaches are not efficient in case when replicated object are large in size like data stored in multimedia systems (audio, video, graphics, animations). The necessity of the resource locking throughout a time of large objects processing, as well as cost of transfers of such objects in extensive networks lead to a significant decrease in the system efficiency or even may cause a system failure.

Multimedia computing requires real-time guaranteed I/O throughput. Although I/O performance of available storage solutions constantly increases, in many cases it is still not enough efficient to meet the high requirement demand of multimedia. Multimedia data requires a high rate of data transaction and the storage space fills quickly.

Layered Data Replication (LDR) [24] is the replication approach which limits replication of all data only to meta-data replication, such as file indexes or pointers to objects. Algorithm of copy management allows to store data objects separately from the information about real location of data, thus it does not require many copies of data. LDR approaches are based on Voting With Witnesses [74] and Voting With Ghosts approaches [92], but in opposite to the originals they do not require external mechanism for concurrency control in transaction processing.

In [96] authors propose master/slave architecture in which a stream of data is broken into smaller pieces and stored temporarily onto different servers. The reliability of data retrieval and consistency in performance depends on the data flow and available bandwidth. Dedicating specific servers to handle a client request and duplicating the data in the background makes not only the system reliable and simple but also ensures that the storage system is evenly filled. By replicating in-coming multimedia stream onto several servers and re-arranging it back to the original order before storing it will solve multi-user access issues.

The database can support multiple qualities by converting data from the original (high) quality to another (lower) quality to support a users query or precompute and store multiple quality replicas of data items. On-the-fly conversion of
2.5 Review of replication techniques usage

multimedia data is very CPU intensive and can limit the level of concurrent access supported by the database. Storing all possible replicas, on the other hand, requires unacceptable increases in storage requirements. The problem of multiple-quality replica selection subject to an overall storage constraint is addressed in [122]. Authors provide heuristic solutions under two different system models: Hard-Quality and Soft-Quality. Under the soft-quality model, users are willing to negotiate their quality needs, as opposed to the hard-quality system wherein users will only accept the exact quality requested.

2.5.4 Data in mobile systems

Since mobile devices used for data processing (mobile phones, TV or mp3 players, notebooks and so on) have become very popular, the popularity of mobile databases has grown very much as well. Very important feature of mobile system is its ability to optimistic data replication, which means that even in case of disconnection of mobile device operations are performed on local data replicas. However, this leads to a very serious problem caused by the necessity of keeping coherent state of data among replicas after switching on of disconnected device, which requires applying conflict resolution approaches in every copy of data.

The main goal is to store appropriate pieces of data locally at the mobile device so that it can operate on its own data, thus reducing the need for communication that consumes both energy and bandwidth. At some point, operations performed at the mobile device must be synchronized with operations performed at other sites. The complexity of this synchronization depends greatly on whether updates are allowed at the mobile device since keeping coherent state of data among replicas after switching on of disconnected device requires applying conflict resolution approaches in all data replicas.

Eager replication approaches based on Read One Write All (ROWA) [5, 88] with 2-Phase Commit protocol (2PC) implemented fulfill each ACID requirement defined in [29] and ensure entire compatibility with 1-copy serializability [30]. However, since these approaches demand of a continuous connection between every replication node, eager replication is not acceptable in mobile environments.
2.5 Review of replication techniques usage

Modified Read One Write All Available approach (ROWAA), in which read transactions are processed by a single node and update transactions are executed at all available nodes, enables high scalability for read-only workloads [34]. However, because it is required to lock distributed resources between replicas, response time of such systems highly increase and also deadlocks appear more frequently [30]. Thus, because of the mentioned limitations, eager replication approaches are not feasible for mobile systems.

As a result lazy replication approaches are most often applied for mobile systems [120]. This replication is usually realized in Master-Slave architecture (primary copy replication) with the main node (master) and other nodes that store copies of data (slaves). Lazy approach allows on data modifications only in the master node, while data reads are managed by the master and can be performed in any node. In such implementation of mobile replication there is always up-to-date copy of data (primary copy) and there is no necessity of the usage of distributed transactions. The most important flaws of such replication systems are requirement of continuous accessibility of the master node and a bottleneck in the highest loaded master node.

To allow concurrent operation at both the mobile client and other sites during disconnection, optimistic approaches to consistency control are typically deployed [83]. Optimistic consistency maintenance protocols allow data to be accessed concurrently at multiple sites without synchronization between the sites in advance. However, this usually results in short term inconsistencies. Such protocols trade-off quality of data for improving quality of service. Recent survey on optimistic replication is presented in [93]. Coda proposed in [94] uses presented in [62] Isolation-Only transactions consistency protocol and is one of the first file systems designed to support disconnections and weak connectivity. With Two-Tier Replication [30] replicated data have two versions at mobile nodes: master and tentative versions. A master version records the most recent value received while the site was connected. A tentative version records local updates. On the other hand, with Two-Layer Transactions [82] transactions which run solely at the mobile host are called weak, while the other transactions are called strict. A distinction between weak copies and strict copies is that in contrast to strict copies, weak copies are only tentatively committed and hold possibly obsolete
values. Bayou [80, 109, 110] is a replicated, weakly consistent storage system designed for a mobile computing environment that includes portable machines with less than ideal network connectivity. It is built on peer-to-peer architecture presented in fig. 2.8 and consists of a number of replicated servers weakly connected to each other. Bayou does not support full-fledged transactions. A user application can read-any and write-any available copy. Writes are propagated to other servers during pair-wise contracts called anti-entropy sessions.

An interesting extension of the current methods is hierarchical caching for the emerging infrastructures of multi-hop wireless networks [83]. Besides the challenges due to mobility, hierarchical caching introduces new complications such as the multiple levels of intervening caches can that create adverse workloads for the caching schemes used at different levels. A hierarchical caching scheme must have the ability to adapt itself, thereby acting synergistically and cooperatively with other caching schemes on mobile peers [12].

2.5.5 Real time data processing

To fulfill the real time systems requirements, currently used distributed industrial systems or Internet services use data replication techniques. Replication enables reaching an adequate efficiency level, as well as increases an accessibility of the
systems. To maximize benefits offered by data replication in real time systems, it is necessary to design a technique of concurrent transaction processing that ensures high efficiency, especially in cases when the system is overloaded.

In a literature one can find lots of information related to tasks ordering in real time systems working with high loads [19, 47, 57]. In [47] there is a proposition of Replica Concurrency-Control for Overloaded Systems algorithm (RCCOS), which was designed for systems with firmly defined time requirements for processing in real time systems using data replication. The approach is based on a usage of parameters associated with each transaction and called Transaction Importance Value (TIV). In a situation that system works under load higher than defined boundary conditions, value of TIV is used to choose the most important transactions which must be continued. Every other transaction is aborted.

RCCOS approach is an extension of Managing Isolation in Replicated Real-time Object Repositories approach (MIRROR) proposed in [121], which delivered concurrency control algorithm for real time systems. Conflict resolution algorithm used in MIRROR approach is based on Optimistic Two-Phase Locking approach (O2PL) [9] designed for systems without requirements of real time processing.

A very serious drawback of RCCOS approach is that aborts and rollbacks lead to interruptions in processing of important transactions, and in a consequence to improper work of the whole system or even its damage.

2.5.6 Spatial data storing

Spatial database management are the systems which are organized in a way which allows on an optimal storing and management of data related to the spatial objects (for instance points, lines and polygons). Common database systems usually store data represented as numbers and chars, while for spatial processing additional functionality is required to support spatial operations like spatial measurement or construction functions.

While replicating spatial objects between remote locations it is necessary to propagate data in a way that ensure coherency of the spatial objects as a whole. If spatial objects are modified in different nodes additional requirements must be taken under consideration. It means that while modification on a spatial object
is performed in one node, different objects might be in spatial relations with this object. Thus, despite of the fact that these object are not locked, these relation must be respected during processing.

Moreover, restrictions related to the spatial objects make difficult usage of traditional replication approaches for them. If pessimistic approaches are used, overall performance of the system decreases significantly because of resources locking (necessity of locking whole objects or at least parts of them). On the other hand, optimistic approaches allow to modify objects in remote locations at once, and afterwards results are consolidated. However, the independent modifications lead to appearance of incoherent states, and systems must be equipped with complex conflict resolution algorithms or even may be forced to rollback long lasting transactions.

In [44] there is presentation of two locking mechanisms, which can be used in spatial database systems to enable parallel processing of modification operations on spatial objects. These mechanisms are Region Locking and Spatial Relationship-Bound Write Locking. Both approaches are based on statement that transactions which modify objects with spatial relations, need to be propagated and managed by 2 Phase Commit protocol extended to the service of spatial relations (Spatial Relationship-based 2 Phase Commit protocol). Authors say that concurrent updates of two spatial objects having spatial relationships should be propagated and cooperated by using an extended 2-Phase Commit Protocol, called Spatial Relationship-based 2PC protocol. However, a very important drawback of this approach is the fact that it uses locking mechanisms and in consequence system efficiency is decreased. [44] authors also presents an overview of practical system implementation on top of an object-oriented spatial database system Gothic.

An example of a Region Lock is shown in fig. 2.9. $D_{Ri}$ are the objects totally contained in $R_i$ which is 1 road segment and 5 properties. The region $R_i$ is defined by users for updating some objects of $D_{Ri}$. Region locking sets weak SIX locks on $D_{Ri}$, which in fact is a lock mode that holds locks on a set of objects in the shared mode and allows the other transactions to acquire exclusive locks on some of the objects.
2.5 Review of replication techniques usage

2.5.7 Replication for High Availability

Disaster Recovery is the process, policies and procedures related to preparing for recovery or continuation of technology infrastructure critical to an organization after a natural or human-induced disaster. Disaster recovery is a subset of business continuity. While business continuity involves planning for keeping all aspects of a business functioning in the midst of disruptive events, disaster recovery focuses on the IT or technology systems that support business functions. Data replication tools presented in this subsection are used to implement disaster recovery solutions.

A system fulfill High Availability demands when it ensures a certain degree of operational continuity during a given period of time. Availability refers to the ability of the users to access the system, while they are submitting theirs new work, changing work previously done, or gathering the results of previous work. If a user cannot access the system, it is said to be unavailable [81]. Generally, the term downtime is used to refer to periods when a system is unavailable.

Data replication for ensuring High Availability of the system is typically realized as one-way replication with a source data disks and destination data disks in different location. Only data in the source database can be modified (read and write operation allowed), while the destination data is not accessible at all, or is accessible only for read operations. In systems with relational databases data
replication based on Hot Standby technology is used to ensure High Availability in these systems. Hot Standby implementations are Oracle Data Guard [70] implementations, or IBM DB2 High Availability Disaster Recovery (HADR) [10]. When the primary database fails, the secondary instance starts to provide service for clients. To start secondary database, Switchover (performed on the user’s demand) or Failover (automatically after crash) operations are available. The main problem in such configurations is that resources of the secondary database are not utilized.

Such products as EMC Symmetrix Remote Data Facility (SRDF) [23] or Veritas Volume Replicator by Symantec [106] provides remote replication for disaster recovery and business continuity. Processes realized on data blocks level provide host-independent data replication to one or more physically separate systems, which allows companies to deliver 7 x 24 x 365 data availability.

Veritas Volume Replicator and EMC Symmetrix Remote Data Facility are available for many different operating systems (Oracle Solaris, IBM AIX, HP-UX, Linux, etc.). DRBD [33] is an open source software for replication and is implemented for platforms Linux based operating system.

2.6 Summary

The majority of the presented in literature approaches designed for data replication in complex, multi-node environments (for instance multi-agent systems) focuses mainly on the fault-tolerance problems within a system and the security of the system. These approaches address the problems related to the communication and interaction between agents, as well as the coordination of the agents.

Data synchronization in distributed, multi-node systems is performed between a large number of remote nodes, thus, it is required to ensure appropriate scalability level of the whole system with data replication. Since data in the distributed systems can be stored in different database management systems, running on variety of operating systems and hardware platforms, it is necessary to enable cooperation between replication nodes in such heterogeneous systems.

Table 2.2 presents the analysis of the data replication approaches in the aspects of theirs possible usage in various data management systems, as well as
provides the review of existing implementations of these approaches. Most of researches related to data replication in distributed multi-node systems focus on ensuring of system fault tolerance within a multi-agent system, or address the difficulties of making reliable mobile agents which are more exposed to security problems. Thus, the main goal of this research is to propose the replication method for the class of distributed systems with large number of nodes that will ensure high level of scalability.

<table>
<thead>
<tr>
<th>Aspect</th>
<th>System example</th>
<th>Replication Approaches</th>
<th>Practical implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactional Data</td>
<td>OLTP Systems</td>
<td>Eager replication approaches, MADIS middleware replication, Optimistic replication, Middleware based replication with Snapshot Isolation, Ganymed for web replication, Replication approaches using Group Communication Protocol, Total order multicast, Epidemic commit protocols, Voting Copy approaches</td>
<td>Oracle Advanced Replication, Oracle Streams, TimesTen In-Memory DB Replication, IBM DB2 Q Replication, IBM DB2 Master Replication, MS SQL Server Transactional Replication, MS SQL Server Merge Replication</td>
</tr>
<tr>
<td></td>
<td>Multi-Agent Systems, OLTP in Distributed Multi-Node Systems, Mobile Transactional Systems</td>
<td>Approaches ensuring fault tolerance and security</td>
<td></td>
</tr>
<tr>
<td>Large amounts of data</td>
<td>OLAP Systems, Reporting &amp; Analytical Systems, Data warehouses</td>
<td>Lazy replication approaches, RepDB* partial replication, Lazy database replication with ordering guarantees, Middleware Consistent database replication, Replication approaches using Group Communication Protocol, Partial Write Operations</td>
<td>Oracle Partitioning, Oracle Snapshot Replication, Oracle Streams, IBM DB2 SQL Replication, PostgreSQL Slony, MySQL Replication</td>
</tr>
<tr>
<td>Multimedia Data</td>
<td>Media Warehouses, Data Warehouses, Streaming Systems</td>
<td>Layered Data Replication, Voting With Witnesses approach, Voting With Ghosts approach</td>
<td>Replication in RDBMS, Replication in filesystem level</td>
</tr>
<tr>
<td>Mobile Data</td>
<td>Systems accessed by mobile devices (mobile phones, TV, mp3 players, notebooks)</td>
<td>Lazy replication approaches, Replication approaches enabling recoverability, Middleware based replication approaches</td>
<td>Pumatech Sync-it Intellisync, Sycl SIM, fusionOne InternetSync, Aether ScoutSync, Replication in RDBMS</td>
</tr>
<tr>
<td>Data in Real Time Processing</td>
<td>Real Time Systems</td>
<td>Replication approaches based on RC-COS algorithm, MIRROR approach</td>
<td>TimesTen In-Memory DB Replication</td>
</tr>
<tr>
<td>Spatial Objects</td>
<td>Systems supporting spatial measurement or construction functions</td>
<td>Spatial Relationship-based 2PC protocol, Replication based on Region locking, Replication based on Spatial Relationship-Bound Write locking</td>
<td>GIS S/W, Gothic</td>
</tr>
<tr>
<td>Replication for High Availability</td>
<td>System in Disaster Recovery Process</td>
<td>One-way replication, Standby replication, Redundant Array of Inexpensive Databases RAIDb</td>
<td>EMC Symmetrix Remote Data Facility, Symantec Veritas Replicator, Oracle DataGuard, IBM DB2 HADR, IBM Logical VM, DRBD for Linux OS</td>
</tr>
</tbody>
</table>

Table 2.2: Aspects of data replication usage
2.6 Summary

The following requirements are defined to gain the desired efficiency and availability of the designed replication method in the distributed multi-node systems:

- reduction or elimination of the locks between remote replicas which causes that the usage of distributed transactions is not a must,
- minimization of the amount of the messages exchanged between replication nodes,
- transaction processing in parallel,
- ensuring high level of scalability,
- portability and easiness of usage in heterogeneous environments.

The aim of this research is to propose the replication method for the class of distributed systems with large number of nodes, that will overcome the limitations of the currently available approaches, and will be suitable for systems in which demand on high level of scalability is a key issue. The approach is to be suitable for systems working in heterogeneous environments based on various platforms, operating systems or database vendors. Especially, the designed approach should be applicable for IBIS multi-agent system, as a representative of the mentioned above class, to support communication and exchange of information among remote agents.
Chapter 3

Theta replication method

The aim of this research is to propose the replication approach for distributed systems with large number of nodes, which would overcome the limitations of the currently available approaches, and would be suitable for systems in which demand on high level of scalability is a key issue. The approach is to be suitable for systems working in heterogeneous environments with various platforms, operating systems or database vendors. The designed approach must be applicable to support efficient accessibility to the shared common data by every node in a distributed system with many remote nodes.

Theta replication method is designed for high availability multi-tier architecture with distributed middleware, which means that in every site of replicated system there is instance of a middleware software cooperating with instance of a database (located in the same site as middleware). The architecture of the data replication system based on Theta replication is presented in the section 3.2, whereas general assumptions for the approach are presented in the section 3.1 of this chapter.

In the proposed Theta replication method the users’ transactions, globally ordered by an independent transaction identifier issuer, are provided to all database instances (replicas) without preserving any order, where they are reassembled into the order in a way that guarantees the same overall result in all replicas. The middleware uses its own concurrency control mechanism, called Conflict Prevention algorithm, which enables transactions to be executed with relatively high
degree of parallelism. Details on transaction processing in parallel are presented in the section 3.3.2.

The conflict resolution algorithm designed for Theta replication method is relatively simple in action, and because of its simplicity and low time complexity, the approach is very efficient. The system is compatible with the most of existing database systems and since the approach is based on the execution of the stored procedures in remote replicas, it can be used in heterogeneous environments with no difficulty [51, 52]. The key component of the new approach is the Conflict Resolution algorithm realized by the middleware software. It determines an optimal order of transactions execution, to ensure data coherence in database copies and to keep identical data in these copies. The proper order of transaction execution is determined regardless of the time and the order of particular transactions appearance in the middleware. After the conflict resolution is performed, non-conflicting transactions are submitted to database concurrently according to the 1-copy serializability theory [5, 49], whilst conflicting transactions have to be executed apart from transactions they are in conflict with. Conflict resolution and Conflict Prevention algorithm are described in detail in the section 3.3.4.

## 3.1 General assumptions and requirements

To gain the desired efficiency and availability of the systems cooperating with data replicated amongst multiple distributed sites, the following general assumptions and requirements for the proposed data replication approach are made:

- **High scalability** level for data replication in transactional, multi-node environments. A scalable online transaction processing system is the one that can be upgraded to process more transactions by adding new processors, devices or storage, and which can be upgraded easily and transparently without shutting it down. The new approach must be applicable for distributed systems with many nodes, therefore it must provide mechanisms that ensure high scalability.
3.1 General assumptions and requirements

- **Transaction processing in parallel.** To gain better performance of the approach it is required to enable parallelism for processing transaction in database.

- **Portability and easiness of the usage in heterogeneous environments.** To ensure data replication in heterogeneous environments special drivers, connectors or gateways are used. The majority of nowadays transactional systems for data replication in heterogeneous environments is realized using the approaches based on Two-Phase Commit protocol [56]. However, these approaches require to lock resources in databases located in all sites, but the time overhead related to blocking nature of Two-Phase Commit protocol significantly decreases the overall system performance. Thus, the proposed approach should offer features that can be used to implement data replication in the systems working over different platforms, systems or databases.

- **Deadlock detection and resolution.** While locking resources, transactional databases can deadlock. However databases offer many different methods for deadlock detection and resolution. Reduction or elimination of the locks between remote replicas causes the overall performance of the replication is improved significantly [35, 36], thus it issue is also considered in this research.

- **Communication** – reduction of the amount of the messages exchanged between replication nodes. The proposed in a literature approaches require that all messages are delivered in the same order to all sites, which can be guaranteed by the group communication systems and total order communication [18, 76]. The obvious benefit of the approach that does not require any special order of the incoming transactions would be a small number of messages exchanged between remote sites and increased system performance, thus the reduction of the communication amount is an important issue.

- **Fault-tolerance** is a computer system property that enables continuous operating in case of a failure of some of its components. It is required
that the proposed approach to be resistant to the failures of particular components of the whole system.

- Easiness of introducing modifications in the implemented system. Replication system should allow relatively simple changes in an application logic and system updates caused by business requirements.

### 3.2 Theta replication architecture

Fig. 3.1 presents general architecture of the system with Theta replication implemented.

![Figure 3.1: General architecture of Theta replication system](image-url)
Theta replication method is realized as a multi-tier architecture with distributed middleware. This means that in every site of replicated system there is the middleware software cooperating with the database replica.

Database and middleware software can be run either on separated machines or on the same one. The approach allows to use different databases in any site of the system, for instance Oracle in sites 1 and 2, PostgreSQL in site 3 and IBM DB2 in sites 4, 5 and 6. Moreover, databases in every site can be implemented to support additional technologies required in particular sites. It means that it might be the single instance database in site 1, clustered database in site 2, and database replication based on Hot Standby mechanism implemented or hardware level replication in site 3.

![Figure 3.2: Communication layers for Theta replication](image)

Clients connect to database throughout middleware which uses appropriate driver for connection with particular database. It can be database native driver.
(OCI for Oracle for instance) or other data access drivers, such as ODBC, JDBC, OLE-DB, etc. Client uses a special driver called Theta Connector, which in fact is a relatively simple and fast software that automatically converts their transactions into the arrays of parameters, and then submits them to the middleware. Communication layers for Theta approach is presented in fig. 3.2

The flow in Theta replication system is presented in fig. 3.3. If transaction received by middleware is a new one, it is forwarded to every other site, otherwise it is processed locally only. Moreover, by default read-only transactions are processed only in a database local to middleware. After receiving transactions, middleware software processes them performing conflict resolution procedure.

Figure 3.3: Data flow in Theta replication system

The user’s transaction is submitted to middleware, where it is forwarded to every other middleware instance. The middleware node which receives the current transaction from the user is called Middleware Transaction Manager. One transaction can be managed only by one Middleware Transaction Manager, however manager is run at every site where the new transactions arrive. Middleware
transaction manager submits the client’s transaction to the local database as a stored procedure call. After transaction processing is finished in database, it receives the status of the transaction execution from the local database, and then Middleware Transaction Manager forwards this status to the user. Middleware node that receives transactions forwarded by Middleware Transaction Manager does not return information to the user – only submits this transaction to the local database and verifies the status of its processing.

System allows to configure direct connections between users and local middleware. Although users are allowed to connect any middleware instance, it is recommended to establish the users’ connections to the middleware nearest to the particular user, which considerably reduces communication load.

3.3 Approach details

This section provides detailed description of Theta replication method.

3.3.1 Middleware components

At each site of the replication system there is an instance of the middleware and an instance of the database. It is also possible that both middleware and database instances work in cluster configuration to increase fault-tolerance and/or performance. The middleware is located between the clients submitting transactions and the database. To submit transaction client uses a dedicated driver called Theta Connector. Theta Connector converts the users’ requests into the straight sets of parameters. Afterwards, these parameter sets are sent to the middleware, where conflict resolution is performed, and then in the middleware parameters are decoded into stored procedures calls and submitted to each copy of the database. It can be simple procedures inserting or modifying single row or very complex procedures operating on many tables. The middleware process executes stored procedures related to one array of parameters as a single transaction.

The middleware (fig. 3.4) consists of the 5 main components which communicate each other and perform actions to guarantee the same results of the
transactions execution in all the remote replicas, despite the fact that transactions are provided to middleware in various order. These middleware components are: communication manager, global identifier generator, queue manager, conflict resolution manager and stored procedure executor.

The connection manager (cman) is the interface that handles the clients’ connections in the middleware. After receiving the client’s transaction, the connection manager appends it to the local transaction input queue and forwards it to every other nodes. It also manages the connections between middleware nodes and communication with the global identifier generator.

The global identifier generator (gidg) provides transactions with a unique sequential identifiers.

The queue manager (qman) is the process which selects transactions with consecutive identifiers from the input queue. These selected transactions are then submitted to the conflict resolution manager.

The conflict resolution manager (crm) is the key component of the proposed replication approach. It maintains the conflict resolution for transactions.
3.3 Approach details

placed in the shared memory and assigns the proper order of the execution of transactions. When the conflict resolution manager marks a transaction to be processed in database it is forwarded to the stored procedure executor.

The stored procedure executor (spex) executes procedures in local database using either native API provided with database systems, or universal database APIs. The transactions which are placed in the input queue of the stored procedure executor, are executed in database in parallel as stored procedure calls.

3.3.2 Transaction processing

The following subsection presents interactions between middleware components and provides steps that are required in the middleware to execute the user’s transaction in database.

Depending on the type of a transaction, specific action is undertaken. If processed transaction is read-only, it is executed locally only, without Conflict Resolution, therefore connection manager passes it immediately to the local database. Otherwise, if the transaction modifies, adds or deletes data, Conflict Resolution must be performed.

The client’s transaction consists of global parameters, procedure names and other related arguments. After receiving the transaction details, connection manager requests the consecutive transaction identifier (TID) from the global identifier generator, and assigns it to the list of the parameters defining transaction. Afterwards this list of parameters is distributed among all the middleware nodes in the system and appended to the input transaction queues in every site. A subset of transactions with the lowest and consecutive transaction identifiers is then chosen from the input queue by the queue manager, and then placed in a shared memory, where it is available by the conflict resolution manager. In the next step the proper execution order for the selected transactions is determined, which is done on the basis of the Conflict Resolution algorithm realized by the conflict resolution manager. Afterwards data in the transaction list of parameters is transformed into the stored procedure calls. These stored procedures are then executed in database copies, using native or standard database API.
3.3 Approach details

Conflicting transactions have to be run apart from transactions they are in conflict with, and the stored procedure executor sends them to database depending on the global transaction identifier. The conflict resolution manager selects the transactions which can be processed concurrently, and these transactions are processed concurrently in the database.

An idea of transaction processing based on Theta approach usage is shown in fig. 3.5. Tasks performed while transaction is being processed are performed by particular middleware processes – the names of the processes that are involved in the presented tasks are included in brackets in figure 3.5.

Figure 3.5: Transaction processing in Theta approach

The algorithm of transaction processing presented in fig. 3.5 consists of the following steps:

1. Transaction submitted by a user is received by middleware CMAN process.
2. CMAN communicates with GIDG process and receives transaction identifier TID.
3.3 Approach details

3. Transaction type is verified (Read-Only, SQL or Stored Procedure) and depending on it the following actions are undertaken.

**IF transaction is Read-Only**

4. CMAN forwards transaction directly to the local SPEX input queue.

5. SPEX reads transaction data and calls an appropriate procedure in local database only – without preserving any order as the transaction is read-only.

6. SPEX receives transaction status or database results and sends them back to CMAN.

7. CMAN returns data or transaction status to the user.

**ELSE IF transaction is in form of SQL query**

4. CMAN forwards transaction to CMAN processes in every other middleware localizations.

5. CMAN forwards transaction directly to local SPEX input queue.

6. Based on transaction identifier, SPEX reads transaction data and execute SQL statement in database.

7. SPEX receives transaction status or database results and sends them back to CMAN.

8. CMAN returns data or transaction status to the user.

**ELSE IF transaction is in form of Stored Procedure**

4. CMAN forwards transaction to CMAN processes in every other middleware localizations.

5. CMAN assigns transaction to local QMAN transactions input queue.

6. From all transactions in the input queue, QMAN chooses a subset that contains transaction with lowest TID and transactions with subsequent TIDs.
7. If selected transactions are not in conflict with others in a subset or has the lowest TID it is inserted into CRM input queue, otherwise transaction is returned back to QMAN.

8. CRM performs Conflict Resolution procedure.

9. Based on Conflict Resolution results CRM submits transactions to SPEX in appropriate order.

10. SPEX reads transaction data and calls stored procedure in database.

11. SPEX receives transaction status or database results and sends them back to CMAN.

12. CMAN returns data or transaction status to the user.

When the transaction forwarded by Middleware Transaction Manager for this transaction) is received by connection manager processes in other site, it is not forwarded any more. To decide whether the transaction is submitted by the user or by other middleware, connection manager uses TID identifier, which is empty for new users’ transactions.

### 3.3.3 Communication in middleware layer

The list of the parameters defining particular transaction consists of the vector containing transactions identifier (TID) and global parameters for transactions, and the array of parameters related to the stored procedures involved in the transaction. A sample data structure consisting list of parameters for a single transaction that is placed in the middleware queue is presented in fig. 3.6. The number of the parameters in this package depends on the quantity of singular operations processed on the rows related to each table involved in the transaction. For instance, if the transaction updates two different rows – one in Clients table and one in Orders table, then two stored procedures connected with these updates must be executed, and the data structure for this message contains two rows for both ClientUpdate and OrderUpdate procedure calls.
3.3 Approach details

The majority of database applications are implemented as the forms using text based controls, numeric based controls, and so on. This causes that Theta Connector interface does not need to perform any additional conversion of data while sending them to the middleware and it’s implementation is not complex and nor difficult. In many cases data types in forms are compatible with variables types in communication module and type conversion is not necessary.

To apply the client’s transaction Theta Connector submits packaged messages with the details of the transactions to the middleware. Detailed format of the message is illustrated in fig. 3.7. Each data structure related to a certain transaction consists of transaction identifier, node identifier, transaction type, global transaction parameters and parameters describing particular operation in database system.
3.3 Approach details

Data structure example

En example of the data structure containing list of parameters is presented in fig. 3.8. The provided transaction is an update transaction (N – data modification, Y – read-only), which is started in node 24 and its TID is 00007232202296330079200560902340778. On the purpose of this transaction storage procedure executor calls UpdateEmployee procedure in database, which updates the employee’s salary based on the employee identifier. The UpdateEmployee procedure is executed 3 times, once for each row in the presented data structure. EmpID (employee identifier) and float variable are the parameters that are used in every call of the UpdateEmployee procedure.

<table>
<thead>
<tr>
<th>Data structure for transactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>UpdateEmployee 80320 4000.00</td>
</tr>
<tr>
<td>UpdateEmployee 9805 7950.07</td>
</tr>
<tr>
<td>UpdateEmployee 80084 11920.60</td>
</tr>
</tbody>
</table>

Figure 3.8: Data structure containing details of transaction

3.3.4 Conflict resolution

When transactions interleave their operations during execution they are concurrent. If concurrent transactions overlap their operations in a way that they use the same data, then conflict appears. Concurrency control is the activity of coordinating an execution of concurrent transactions which may interfere with each other. Transactions without conflicts do not interfere each other as they do not interleave their operations, therefore concurrency control is only relevant to concurrent conflicting transactions.

In Theta replication method, at the bottom of conflict resolution is a fact that transactions which are not in competition for system resources with each other, do not have to be processed in the same order in all replicas but they can be processed concurrently. Based on this statement, the designed mechanism
provides a scheduler for transactions that orders transactions in a way allowing
them to be executed concurrently without preserving any order.

When a transaction is delivered to a site connection manager associates it
with the subsequent transaction identifier, and adds it to the input queue. Queue
manager uses TIDs selection (transactions selection) to prepare an initial subset
of input transactions that are further processed by conflict resolution manager.

**TIDs Selection details**

Let:
- TID be the transaction identifier,
- \(Q_i\) be the transaction input queue elements at time \(T\),
- \(Q_s\) be the subset of transactions chosen from the input queue,
- \(N\) be the number of transactions in the input queue,
- \(L\) be the lowest TID for each elements actually in \(Q\),
- \(M\) be the maximum TID for each elements actually in \(Q\),
- \(P\) be the maximum TID from the previous iteration of transaction selection, \(P\)
defaults to 0,
- \(G\) be lowest TID for transaction not yet received but satisfying the condition:
  \(P < G < M\).

Assuming that each subsequent transaction arrived middleware with preserv-
ing the TID order, the transaction input queue contains the following elements:

\[
Q_i = \{q_{P+1}, q_{P+2}, q_{P+3}, q_{P+4}, \ldots, q_{N-P-1}, q_{N-P}\} \quad (3.1)
\]

Since transactions are received without preserving any order, at time \(T\) the
transaction input queue may contain any of the following elements:

\[
Q_i = \{q_{P+L}, q_{P+L+1}, q_{P+L+2}, q_{P+L+3}, \ldots, q_{N-P-L-1}, q_{N-P-L}\} \quad (3.2)
\]

In the process of transactions selection (TIDs Selection), queue manager sorts
out elements of \(Q\) using Quick-Sort algorithm \[102\], and then chooses the subset
of \(Q\) containing the following elements:
3.3 Approach details

\[ Q_s = \{q_{P+L}, q_{P+L+1}, \ldots, q_{N-P-G-1}\}, G > P + 1 \]  \hspace{1cm} (3.3)

\[ Q_s = \{\emptyset\}, G <= P + 1 \] \hspace{1cm} (3.4)

It means that for \( G > P + 1 \) \( Q_s \) consists of transactions with continuous, subsequent TIDs, starting with TID = \( P + 1 \). If \( G = P + 1 \) no transaction is selected in the iteration since there is no subsequent transactions in the queue that can be processed (still exists gaps in input queue and preceding transactions need to be received. Transactions with \( G < P + 1 \) can not be processed in the current iteration since the iteration starts with TID = \( P + 1 \) (the lowest currently processed TID) and no transactions is chosen.

**TIDs Selection example**

For better understanding of TID selection an example of TID selection process is shown in fig. 3.9.

Transactions in the middleware input queue are identified by TIDs, which is shown in fig. 3.9a. In the first step transactions from the input queue are copied to a temporary vector used in selection algorithm (fig. 3.9b). Afterwards, Quick-Sort algorithm is used to sort out elements (fig. 3.9c). Finally, the first subset of subsequent TIDs is chosen and forwarded to conflict resolution manager input queue (fig. 3.9.d).
3.3 Approach details

Conflict Resolution details

Conflict resolution for Theta replication method is realized by the middleware CRM process. Activity diagram for Conflict resolution is presented in figure 3.10.

When the selection of a subset of transactions (TIDs) considered for being submitted to the database is done, the conflict resolution procedure for those selected TIDs is performed. As it is presented in fig. 3.10, each transaction is taken under consideration, and in the loop it is compared with every other transaction in case of possible conflict appearance. If a subset of transactions is not in conflict with other transactions, these transaction are passed to SPEX and then executed in parallel in database. Otherwise, if two transactions are in
conflict with, the transaction with the lower TID is submitted to database, while the other transaction is returned back to CRM input queue for further processing.

In the first iteration of the algorithm the transaction with the lowest TID is taken under consideration. In each step of the first iteration two parameters of the transaction with the lower TID (procedure name and processed row ID) are compared with the same parameters of the rest of transactions chosen from the conflict resolution manager input queue.

In the next iterations consecutive transactions from the queue are compared with transactions with greater TID, until all selected transactions are not examined for conflict resolution. Transaction TID\(_i\) is compared with all subsequent transactions \(\{\text{TID}_{i+1}, \text{TID}_{i+2}, \ldots, \text{TID}_N\}\), then the transaction with TID\(_{i+1}\) is compared with all subsequent transactions \(\{\text{TID}_{i+2}, \text{TID}_{i+3}, \ldots, \text{TID}_N\}\), and so on.

If subsequent transactions chosen from the queue of transactions are not in conflict with each other, all these transactions are submitted in parallel to the database (after completion of all preceding transactions with TIDs lower than the lowest TID for currently processed transactions). Furthermore, if the transaction is not in conflict with all of the preceding transactions, it is submitted to the database immediately, even though there are some conflicting transactions with smaller TIDs in the input queue. According to the serializability theory [5] non-conflicting transactions are executed concurrently ensuring data is coherent in all replicas regardless of the order of transactions execution. On the other hand, if two compared transactions have the same row ID for at least one procedure name, transaction with higher TID is marked as conflicting transaction and can be processed after preceding transaction is completed.

**Conflict Resolution example**

Fig. 3.11 presents an example of conflict resolution. In the example three transactions with TID 99, 100 and 101 are considered. In the first iteration of the algorithm the transaction with the lowest TID is taken under consideration \((L = 99)\). In each step of this iteration two parameters of the transaction with lower TID (procedure name and processed row ID) are compared with the same parameters
3.3 Approach details

Figure 3.11: Idea of Conflict Resolution

of the rest of transactions from the conflict resolution manager input queue (100 and 101).

In the next iteration consecutive transactions from the queue are compared with transactions with greater TID, until all selected transactions are not examined for conflict resolution (transaction 100 is compared with transaction 101 in the example). In this example the transaction with TID = 99 is in conflict with transaction with TID = 100 since it contains the procedure named ProcName2 with row ID equal to 77, and the same set of parameters is present for the transaction 100. The transaction 99 is therefore submitted to database immediately. The transaction with TID = 100 is marked as conflicting transaction and is processed after the transaction 99 is completed.

The transaction with TID = 101 is submitted to database before the conflicting transaction 100. In fact it can be submitted at once with the transaction 99 without being influenced by the conflicting transaction 100.

3.3.5 Failure resistance

One of the goals of the data replication is fault tolerance - if certain replica crash, the database can still be available. This section provides the issues related to the fault tolerance that can be obtained in the system with replication based on Theta method.

<table>
<thead>
<tr>
<th>ProcName1</th>
<th>1</th>
<th>ProcName2</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcName1</td>
<td>2</td>
<td>ProcName2</td>
<td>77</td>
</tr>
<tr>
<td>ProcName2</td>
<td>77</td>
<td>ProcName3</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProcName1</th>
<th>4</th>
<th>ProcName2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcName2</td>
<td>77</td>
<td>ProcName3</td>
<td>1</td>
</tr>
</tbody>
</table>
3.3 Approach details

Transaction log

*Connection manager* can work in 2 modes – with transactions archiving enabled or disabled. When archive mode is enabled all transactions processed by CMAN are logged into a special transaction log. Transaction information is registered into this log before it is submitted to the other replicas, and before the transaction is processed in the database. The idea of transaction logging for in middleware database recovery process is presented in figure 3.12.

![Figure 3.12: Middleware transaction logging](image)

Failure scenarios and recovery process

Transaction logs are used in database recovery process to recover data after a system failure. The following are the possible failure scenarios considered for the system with Theta replication implemented.

- one replica fails, related middleware still can handle connections,
3.3 Approach details

- middleware fails, related replica works properly,
- both middleware and replica fail.

In a case that one replica fails and its related middleware still can handle connections, middleware processes cooperating with the failed replica CMAN behaves as a client of CMAN in other location. Middleware CMAN works as a bridge that passes client transactions to other middleware and after receiving transaction status from the middleware passes the status back to the client. CMAN also logs transactions in its transaction log for further database recovery.

Figure 3.13: Idea of transaction recovery inside middleware
After only a middleware fails, the middleware location is excluded from the replication. When middleware is brought back to operating state, Recovery Manager process (RCVMAN) requests remote CMAN for current transaction log, and then applies changes to database. RCVMAN submits transactions to database through QMAN process, which is presented in fig. 3.13. This ensures that Conflict Resolution is performed for transactions in a normal way, and therefore data replica is in consistent state and is synchronized with other replicas. After recovery process if finished, CMAN starts to handle connections.

When middleware and replica fail at the same time, first database is restored using database restoring procedures appropriate for particular database vendor and version. Then the database is brought to a consistent state using database on-line logs (transactions archive logs). After database recovery is finished, middleware Recovery Manager process verifies transaction log to find transactions that need to be run in database to achieve data synchronization with other replicas. RCVMAN process reads transaction log and applies appropriate transactions to the local replica. Finally, after middleware recovery is finished, CMAN starts to handle clients connections and normally processes transactions.

### 3.4 Complexity analysis

In the Conflict Resolution algorithm implemented for the proposed approach sorting and comparing of simple arrays of variables are performed. All of these operations are based on the usage of the fastest available algorithms [14]. The Quick-Sort algorithm is used for TIDs Selection and for Conflict Resolution.

The following sections presents the time complexities for the selection of the transaction identifiers and for the conflict resolution.

#### 3.4.1 Selection of the transaction identifier

Let $n$ be the length of the transaction queue (the number of transactions in the queue).

The first step of the selection of transaction identifiers is to sort out all the elements from the input queue using Quick-Sort method, which complexity is
θ(log(n)). Then the lowest, subsequent elements from the queue are selected. The complexity of this operation is equal to θ(n), and the overall time complexity of the TIDs Selection is given by the greater of these two complexities:

\[ \theta(n \log(n)) + \theta(n) = \theta(n \log(n)) \]  

(3.5)

### 3.4.2 Conflict resolution

Let:

- \( p \) be the number of transactions chosen from the queue of transactions,
- \( n_1, n_2, \ldots, n_p \) – numbers of single operations/procedures per transaction (numbers of rows in the list of parameters related to the processed transactions).

In the first step of finding possible conflicts between \( p \) selected from the queue transactions, the parameters related to this transactions are sorted out. It requires \( p \) sorts (one sort per each set of parameters) using the Quick-Sort algorithm. The time complexity of all these sorts is expressed as below:

\[ \theta(n_1 \log(n_1)) + \theta(n_2 \log(n_2)) + \ldots + \theta(n_p \log(n_p)) \]  

(3.6)

Expression (3.6) for \( n = \max\{n_1, n_2, \ldots, n_p\} \) is equal to:

\[ p \cdot \theta(n \log(n)) = \theta(n \log(n)) \]  

(3.7)

Since constant factors can be ignored in left side of equation (3.7), the time complexity of sort operations for lists of all parameters is equal to \( \theta(n \log(n)) \).

When the parameters in each list of transactions are sorted out, they are compared \( n_i \) times, where \( n_i \) is the greater number of rows in both lists of parameters. The time complexity of the comparison of two sorted list is equal to \( \theta(n_i) \). The time complexity of all comparisons is the aggregate of the complexities for single comparisons. After ignoring constant elements and for \( n = \max\{n_1, n_2, \ldots, n_p\} \) it is equal to \( \theta(n) \), which is presented in the equation (3.8).

\[ (p - 1)\theta(n) + (p - 2)\theta(n) + \ldots + (p - (p - 1))\theta(n) = \theta(n) \]  

(3.8)
3.5 Summary

The overall time complexity of all the operations required to perform the conflict resolution is equal to:

\[ \theta(n \ast \log(n)) + \theta(n) = \theta(n \ast \log(n)) \] (3.9)

3.5 Summary

Theta approach is realized in multi-tier architecture with distributed middleware, which gives mechanisms that ensure high level of system scalability. Since Theta replication method does not require any distributed locks and transactions are processed in parallel only if they are not in conflict, deadlock detection and resolution is realized using the same techniques as in the centralized database system. Elimination of the necessity of the usage of the locks between remote replicas causes that the usage of distributed transactions is not required, and in consequence the overall performance of the data replication improves significantly.

The Conflict Prevention algorithm designed for the approach does not require any special order of the incoming transactions, which minimize amount of messages exchanged between remote sites, considerably increasing the overall performance of the replication process.

The clients' transactions are processed concurrently in database which allow to gain better performance of the replication process. The great advantage of the proposed approach is the usage of Conflict Prevention algorithm in conjunction with executing of procedures stored in databases. The new approach uses innovative mechanism determining the order of transactions execution, which allows to apply transactions in replicas concurrently. The other advantage of the approach is the fact, that on behalf of middleware stored procedures are executed in database. These stored procedures are precompiled in the database kernel, thus they are more efficient than execution of single SQL instructions.

System components and stored data in the system with Theta replication implementation are duplicated among different sites, overall system resistance to failures of a single component or even a whole site fulfill defined data safety requirements.
3.5 Summary

In Theta replication stored procedures are executed in databases using native drivers in middleware layer for communication with particular database, which eliminates necessity of modifications in the application code to fulfill particular database requirements. Since procedures in various databases use identical names in each replica and exactly the same list of parameters, it is possible to use different platforms and various databases. Moreover, changes in the application logic caused by business requirements do not require modification in the code of Theta replication core system. Similarly to applications working with single instance database, only the client’s application and database objects have to be modified. In the middleware it is only required to redefine relations between new and existing stored procedures.

To summarize, Theta approach uses both adapted components appearing in literature or implemented in practice solutions (for instance global transaction identifiers, distributed middleware architecture), and the new elements (Conflict Resolution algorithm, the way of communication between processes inside the middleware and remote sites). Thank to its features, the new approach offers a highly scalable, efficient and failure resistant data replication technique for distributed, multi-node systems. The new method is suitable for environments working on various hardware platforms, with different operating systems and databases.
Chapter 4

In-laboratory testing

The main purpose of the laboratory experiments is verification of Theta replication method presented in chapter 3.

The following general assumptions and requirements introduced in 3.1 are defined for Theta replication method:

- high scalability level of data replication in transactional, multi-node environments,
- transaction processing in parallel,
- portability and possibility of the usage in heterogeneous environments,
- deadlock detection and resolution,
- reduced amount of communication,
- resistance to failures,
- easiness of introducing modifications.

The desired way of Theta replication scalability evaluation would be a comparison of the method with other approaches, for instance [17, 54, 61, 76, 77, 84]. However, setting up of the environments identical to those used in preceding tests is not possible because of the limited availability of the hardware in configurations used within tests of previous methods. Moreover, the source codes and deployment software of the replication approaches are not available, which in fact not
allow to arrange such test environment that would allow to conduct authoritative comparison of the approaches. Because of these issues, the majority of the replication approaches evaluations presented in literature are also not based on the available benchmarks but they are done as valuation of the approaches features in case of theirs particular usage. Thus, also Theta replication method evaluation is performed in a dedicated test environment, which allows to verify whether it is applicable for the certain solutions that it is designed and implemented for.

As it is presented in chapter 3 the usage of Conflict Prevention algorithm in conjunction with executing of procedures stored in databases allows on concurrent transaction processing when Theta replication method is used. Additionally, mechanisms used in Conflict Prevention process prevents from conflicts appearance on the level of a single instance of the database management system (single replica), which means that transactions cannot deadlock, thus deadlock resolution is not required. Finally, Theta method is designed in a way that requires transaction to be processed in a node local to client. This node processes transaction and simultaneously forwards transaction to every other nodes. However, no confirmation whether transaction is committed in remote nodes, thus only a small number of messages exchanged between remote nodes.

It is decided that fault tolerance of the method and easiness of introducing modifications in systems based on Theta method are explored within real life evaluation, which is presented in chapter 5.

To sum up, the aim of the experiments performed in the laboratory environment is to explore the portability and scalability of Theta replication method.

The first section of this chapter covers the issues related to the testing implementation of Theta replication method. Then there is an introduction to the performance indices used within experiments and presentation of the testing plan. Finally, the results obtained during the experiments performed in laboratory environment with Theta replication method implementation are presented with theirs extensive analysis.
4.1 Testing implementation

General architecture of the replication system based on Theta replication method is presented in fig. 3.1. Implementation of Theta replication approach requires the following requirements to be satisfied:

- Data replication is totally transparent for clients. This means that replicas for clients are visible as regular databases.

- Replication is organized as a set of components realizing functionalities ensuring data in replicas is coherent and up to date.

- Data replication modules are seen as a 'black box' that is placed between clients and replicas and realizes appropriate functionalities according to replication needs.

- Possibility that replication modules can be run on dedicated machine or placed on the machine running database engine.

Fig. 4.1 presents general location of Theta replication method implementation in the overall data replication system. Presented in figure replication modules behave as 'black box' from the user’s point of view. According to Theta replication architecture presented in 3.1 replication modules are implemented as middleware components located between clients and replicas.

Middleware software in Theta replication approach consists of five components presented in 3.3.1. Three middleware components \((GIDG, QMAN, CRM)\) are used as internal mechanisms of replication method. These three middleware components support additional two components, \(CMAN\) and \(SPEX\), which are used for communication with external environments. This implies implementation of \(CMAN\) and \(SPEX\) components need to fulfil the additional demands.

Requirements for \(CMAN\) implementation:

- working as a server in client/server architecture,

- running a listener waiting for clients connection,

- handling clients connections in parallel threads,
4.1 Testing implementation

- managing clients sessions, which includes establishing connections and exchanging messages between clients and middleware,
- forwarding transactions among every node in replication environment,
- providing clients with the status of the executed transactions or with data returned from database.

Requirements for SPEX implementation:
- reading of transactions data from SPEX input queue,
- converting data from the SPEX input queue into storage procedure call,
- executing procedures in related local data replica and receives database response,
- allowing on configuration of parameters for database connections – database drivers, server ip, database account and password defined in external configuration file(s).
4.2 Performance indices and testing plan

According to these system and processes requirements fig. 4.2 presents graphical explanation of middleware interfaces to the external world. Presented schema shows the middleware components and theirs outgoing communication mechanisms in only one location (single middleware instance) which is a part of the whole replication system.

Figure 4.2: Middleware external interfaces

4.2 Performance indices and testing plan

The following section presents the main performance indices used within Theta replication method evaluation and then provide plan of the tests.
4.2 Performance indices and testing plan

Scenarios

The following part of the section introduces a notation for scenarios defined in this research. A scenario is represented by a 6-elements collection presented in table 4.1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Centralized database or Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>Number of replicas</td>
</tr>
<tr>
<td>Size of tables</td>
<td>The size of the tables in each database copy defined as number of rows in table</td>
</tr>
<tr>
<td>Load</td>
<td>Amount of transactions per second</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>The percentage of queries for all transactions</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>The percentage of conflicting transactions among all transactions [%]</td>
</tr>
</tbody>
</table>

Table 4.1: Scenario definition

The notation in the example scenario 4.2 means that S is a scenario in which Theta replication method is used, and the system is realized with 2 replicas. The size of the tables in each database copy is set to 10 000 rows. The total number of clients is 40 and each client submits to database 1 transaction per second, which means that the users’ load is equal to 40 tps. The amount of queries in the set of all submitted transactions is 0% and 20% of all submitted transactions are conflicting transactions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10 000 [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>40 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>20 [%]</td>
</tr>
</tbody>
</table>

Table 4.2: Scenario example
4.2 Performance indices and testing plan

Basic information

For a client the time of processing of some Operation o in the database system is the time between the Operation o is submitted by the client and the result of this Operation is obtained by the client. Thus, the time that is spent in multiple middleware and database copies is not important for the client. Therefore, the definitions in this section only take into account the time of the middleware processing of the Operation o, related to its receiving, processing and executing in the database to get the result that the client received.

\( O_{midd} \) – the amount of time that Operation o spends in the middleware from the moment o is submitted by the client until the client receives its result. This time is related to the middleware processing of o, which includes the time when conflict resolution for o is performed and the time when o is executed in the database as a stored procedure call.

\( O_{net} \) – the amount of time that Operation o spends in the network component from the moment o is submitted by the client until the client receives its result. This time is the sum of the time of the delivery of Operation o to the originating middleware (time of communication between client and the nearest replica while submitting o) and the time it takes for the result of o to be sent back from the middleware to the client.

\( O_{total} \) – the amount of time between the moment when o is submitted by the client until the moment the client receives its result. This value is equal to:

\[
O_{total} = O_{midd} + O_{net}
\]  

(4.1)

Mean response time for committed transactions

The response time for a committed transaction can be defined as the time between the submission of the first Operation in the processed transaction and the time at which the client receives the response confirming that this transaction is committed.

Let O be the set of all Operations that are submitted on behalf of transaction T. The response time for a committed transaction T is then formally defined as follows:
4.2 Performance indices and testing plan

\[ T_{total} = \sum_{o \in O} O_{total} \]  \hfill (4.2)

Let \( CT_R \) be a collection consisted of all transactions committed in a particular set of runs \( R \). The \textit{mean response time for committed transactions} performance index is defined as follows:

\[ R_{mean} = \frac{\sum_{T \in CT_R} T_{total}}{\#CT_R} \]  \hfill (4.3)

\textbf{Throughput}

The \textit{throughput} of the system represents the number of transactions that the system commits per a time unit. The throughput can be converted directly from the mean response time for committed transactions. If \( \#CLI_R \) denote the number of clients in a run \( R \), then the throughput of the system is equal to:

\[ \text{throughput}_R = \frac{\#CLI_R}{R_{mean}} \]  \hfill (4.4)

The throughput is an important performance index in the industry benchmarks such as TPC [113]. Since the throughput takes into account the number of clients in the system, it is used in this research in case when the number of clients is different among the particular scenarios. If the number of clients is the same in every compared scenario, the mean response time for committed transaction is used.

\textbf{Network and middleware delays}

Let \( O \) be the collection of all Operations that are submitted on behalf of transactions \( T \). Similarly to the response time, the network delay and the middleware processing delay for a committed transaction are defined respectively as:
4.2 Performance indices and testing plan

\[ T_{net} = \sum_{o \in O} O_{net} \quad (4.5) \]

\[ T_{midd} = \sum_{o \in O} O_{midd} \quad (4.6) \]

Performance indices motivation

A disadvantage of these indices is that they do not include information on the distribution of the measured values. For instance, in the interactive applications with impatient clients, the maximum required response time might be more useful than the mean response time.

However, the mean transaction response time and the throughput are chosen as the main performance indices, since both of them are very often used in either research or industry benchmarks, and both of them take into account all measured values.

Transaction parameters

Clients submit their transactions to the replication system. Transactions are grouped into operations, and within the tests the performance of the transaction processing is measured. Both the kind and the number of submitted operations depend on the type of the transactions which are realized by particular operation. The transactions are characterized according to the following factors:

- **Number of operations per transaction** which is highly dependent on the type of application. For instance, in the simple application enabling money retrieve from an ATM only a few read and write operations are needed, whereas in a decision support system millions of database items may be retrieved from the data files to perform a desired report.

- **Number of transactions submitted per time period** In practice the number of transactions in a period of time delimits the maximum load that the system can handle.
4.2 Performance indices and testing plan

- **Fraction of write operations in update transactions** The number of times that a data operation is a write operation instead of a read operation within the limits of transaction. The fraction of write operations cannot be chosen lower than one per transaction because this would imply that the transactions would be a query (read-only).

- **Percentage of committed transactions** The number of times that a transaction ends with a commit operation instead of an abort operation.

- **Percentage of queries** The number of times that a transaction does not contain write operations.

- **Percentage of conflicting transactions** The amount of conflicting transactions in a set of all transactions.

**Testing plan**

In-laboratory test environment simulates a computing environment where a population of users executes transactions against a database. Organized similarly to the popular TPC benchmarks [113] tests scenarios within this evaluation are centered around the exploration of the scalability and verification of the portability of Theta replication method.

The mean transaction response time and the throughput are chosen as the main performance indices for the scenarios set up within this evaluation. The test scenarios are realized for changing values of parameters characterizing transactions. The performance indices are measured within these scenarios and then appropriate mean values are calculated. The remaining part of this section provides further explanation of the scenarios for Theta method scalability and portability, while scenarios related to the failure resistance and verification of the correctness of system functionalities are presented during real life evaluation presented in chapter 5. The values of the performance indices within the experiments are calculated as the mean values of the results gained within 5 runs of the experiment.
4.2 Performance indices and testing plan

Scalability

In telecommunications and software engineering, scalability is the ability of a system, network, or process, to handle growing amounts of work in a graceful manner or its ability to be enlarged to accommodate that growth [6]. For example, it can refer to the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added.

Scalability can be measured in various dimensions [119], such as:

- Load scalability which is the ability for a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads. Alternatively, the ease with which a system or component can be modified, added, or removed, to accommodate changing load.

- Geographic scalability – the ability to maintain performance, usefulness, or usability regardless of expansion from concentration in a local area to a more distributed geographic pattern.

- Administrative scalability defined as the ability for an increasing number of organizations to easily share a single distributed system.

- Functional scalability – the ability to enhance the system by adding new functionality at minimal effort.

In the systems with the data replication implemented additional replicas increase the availability of the system, which is confirmed by many preceding researches [54, 84, 114]. However, the communication overhead usually increases and additional database copies may cause decrease in the overall performance of the system. On the other hand, in some cases the performance can be even improved when new replicas are added.

To verify the scalability properties of the data replication process based on Theta replication method the following issues influencing data replication performance are considered:

**Various numbers of replicas.** First of all, scalability tests for Theta replication methods means it is tested with the various number of the replicas, including
4.3 Results

the experiments in which comparison of Theta replication method and centralized database is made.

**Percentage of reads.** The percentage of queries is an important factor when replication technique is evaluated. This parameter characterizes the behavior of the approach when the application needs to process many queries, many update transactions or both of them. The percentage of read operations is examined in the scenario with the ratio of read to write transactions varying among the subsequent runs.

**Conflicting transactions ratio.** Similarly to the percentage of queries the percentage of conflicting transactions is the replication parameter that significantly influences the replication system behavior. The ratio of conflicting to non-conflicting transactions in a set of all submitted to database transactions considerably depends on the kind of application. In IBIS system, like in the majority of database systems used for real world applications, the range of conflicting transactions floats around 10% to 30% of conflicting transactions among all transactions. Therefore 10, 20 and 30 ratios are used in the method evaluation.

**Size of database copies.** To evaluate the influence of the time of transactions on the data replication process the experiments with different size of the database copies are performed.

**Portability**

Portability and easiness of the usage of Theta replication method in the heterogeneous environments is also examined. Theta replication implementation is verified for SUN Solaris, Linux RedHat and Windows XP operating systems, which are run on different hardware platforms – SPARC, AMD, Intel. Oracle and PostgreSQL are the relational database management systems involved in tests.

4.3 Results

The factors influencing the replication process are considered in the following tests cases realized within the research. First, there is an extended discussion of the scalability of the approach, which is tested with the various number of
4.3 Results

the replicas. Scalability tests include also the experiments in which comparison of Theta replication method and centralized database is made, which gives a general characteristic of the approach from the point of view of the scalability. Afterwards, there is a discussion of the influence of the percentage of queries in submitted transactions on the replication technique. Following this there is an evaluation of the impact of the percentage of conflicting transactions on the approach. Finally, the replication method is tested with different amount of data in the replicated databases with and without indexes.

The test environment organization with the specification of the hardware used is presented in the appendix C.

4.3.1 Scalability

Additional replicas increase the availability of the replication system, which is confirmed by many preceding researches [54, 84, 114]. However, in such systems the communication overhead usually increases, and extending system with additional database copies may cause decreases in the overall performance of the system. On the other hand, in many cases the performance is improved when new replicas are added. Thus, this issue is extensively examined in the following subsection, where there is an evaluation of the scalability of of Theta replication method. Scalability of Theta approach is the crucial factor in the evaluation of the proposed approach.

**Various numbers of replicas**

Theta approach scalability analysis means that the system is examined from the point of its behavior when the number of replicas changes. Replication scalability tests are performed for the scenario presented in table 4.3 with 1 database copy and 2, 5, 10 replicas. Both 1 copy database and replicated systems used in this scenario are based on the implementation of Theta replication method. The load changes from 5 up to 100 tps and 100% of submitted transactions are modifications, 20% of which are conflicting transactions.

As it is shown in fig. 4.3, the mean response times of the replication system are close to the response times of the replication system built with two database
4.3 Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>1, 2, 5, 10</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10 000 [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>5-100 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>20 [%]</td>
</tr>
</tbody>
</table>

Table 4.3: Scalability scenario with various numbers of replicas

copies only. It is also easily noticeable that in a case with 1 database replication implementation mean response times are slightly lower than for the system with at least 1 copy. On the other hand, it is worth to notice that for 2 replicas or more mean response times are very similar to the results gained for the system with two replicas.

![Various numbers of replicas](image)

Figure 4.3: Scalability
4.3 Results

The reason of such results is that the same sets of transactions are executed in every database in parallel. Only conflicting transaction execution is performed in a serial way. Moreover, the overhead for communication between middleware and database is very small since Oracle Call Interface\(^1\) (OCI) library is used for implementation. Middleware communication overhead is also reduced by exchanging only short messages between middleware components, without waiting for transfer receiving confirmation. Actually, for 5 and 10 replicas replication system behaves similarly to the system with 2 replicas, as no additional time is required to exchange messages between replication nodes (1 transaction is related to exactly 1 message which is forwarded to every other node in parallel). For more database copies assigned to the replication system a communication overhead to the response time is caused only by the socket maintenance in the operating system, which can be noticed for the loads higher than 80 tps.

<table>
<thead>
<tr>
<th>Load [tps]</th>
<th>1 Replica</th>
<th>2 Replicas</th>
<th>3 Replicas</th>
<th>4 Replicas</th>
<th>Centr. DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>54.8</td>
<td>57.0</td>
<td>56.0</td>
<td>58.4</td>
<td>8.6</td>
</tr>
<tr>
<td>10</td>
<td>55.8</td>
<td>61.0</td>
<td>60.0</td>
<td>60.2</td>
<td>16.4</td>
</tr>
<tr>
<td>20</td>
<td>60.2</td>
<td>64.4</td>
<td>63.2</td>
<td>64.2</td>
<td>31.7</td>
</tr>
<tr>
<td>30</td>
<td>60.4</td>
<td>66.0</td>
<td>67.2</td>
<td>67.2</td>
<td>47.0</td>
</tr>
<tr>
<td>40</td>
<td>65.0</td>
<td>71.2</td>
<td>74.2</td>
<td>76.6</td>
<td>67.7</td>
</tr>
<tr>
<td>50</td>
<td>74.6</td>
<td>79.2</td>
<td>82.2</td>
<td>80.6</td>
<td>79.4</td>
</tr>
<tr>
<td>60</td>
<td>77.6</td>
<td>87.6</td>
<td>90.8</td>
<td>94.6</td>
<td>98.3</td>
</tr>
<tr>
<td>70</td>
<td>83.8</td>
<td>93.8</td>
<td>97.4</td>
<td>98.2</td>
<td>109.1</td>
</tr>
<tr>
<td>80</td>
<td>90.0</td>
<td>98.8</td>
<td>106.6</td>
<td>111.4</td>
<td>131.3</td>
</tr>
<tr>
<td>90</td>
<td>94.0</td>
<td>106.6</td>
<td>113.6</td>
<td>121.8</td>
<td>146.2</td>
</tr>
<tr>
<td>100</td>
<td>99.8</td>
<td>116.6</td>
<td>124.4</td>
<td>142.6</td>
<td>169.2</td>
</tr>
</tbody>
</table>

Table 4.4: Scalability test results

Furthermore, the Conflict Prevention in each instance of the middleware is performed for the same number of transactions and does not depend on the number of the nodes in the system implementation. Each transaction at first is processed locally and then is passed on to the remaining nodes in parallel. To commit a transaction no confirmation from remote replica is required, therefore

\(^1\)OCI is a library that supplies programming interface to Oracle database
4.3 Results

the necessity of locking global resources is eliminated and in consequence the amount of communication between replicas is significantly reduced. Thus, the response can be returned to the requesting client almost immediately after the transaction is applied to the local database copy only (database local for the respective middleware within the implementation of Theta method).

For loads defined for the scalability evaluation scenarios, mean response times [ms] are presented in table 4.4. The results gained for configurations with data replication implemented are provided respectively in columns with 1–4 replicas in headers, while results for centralized database are presented in the last column of this table.

Centralized database versus Theta replication

This part of the chapter provides analysis of efficiency of Theta replication method comparing to the centralized database usage. In the Theta Replication scenario, defined in table 4.5, system is implemented with Theta replication and is realized with 1 and 2 copies. 1 copy means a single instance database which works on the basis of Theta replication method.

<table>
<thead>
<tr>
<th>Name</th>
<th>Values ( Theta Replication )</th>
<th>Values (Centralized DB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
<td>Centralized database</td>
</tr>
<tr>
<td>Replicas</td>
<td>1, 2</td>
<td>-</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10 000 [rows]</td>
<td>10 000 [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>5-100 [tps]</td>
<td>5-100 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0 [%]</td>
<td>0 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>20 [%]</td>
<td>20 [%]</td>
</tr>
</tbody>
</table>

Table 4.5: Scenario – Theta replication versus centralized database

The size of tables in each database copy is set to 10 000 rows, the total number of clients changes from 5 up to 100, and the amount of queries in the set of all submitted transactions is 0%. The same conditions as for replicated system are set for the scenario with centralized database with the exception of the database, which obviously is only a single instance database accessed by JDBC. Tests for the scenario with centralized database are performed with the usage of Apache
JMeter 2.3.2 [45] The scenario for centralized database is also presented in table 4.5.

In fig. 4.4 there is a comparison of the mean response times for the centralized system (JMeter graph) and for the implementations of the replication system based on Theta method.

The results for the centralized database and Theta replication method provide an information that for the loads less than 30 tps the average response times for centralized database are shorter than for system using Theta approach, both for a system with and without replication. In this case both the overhead of the communication and Conflict Prevention algorithm performed in the middleware causes the longer response times for the replication system. When the load exceeds 40 tps, the response times for the centralized database are more or less the same as for the replication system. Such situations takes place because the
overhead related to the processing in the middleware is counterbalanced by the distribution of the load among all of the replicas.

Starting with the loads higher than 50 tps the response times for the replicated system are getting shorter than for centralized database system. The explanation of this is the above mentioned load distribution among replicas and the way of session maintaining. In the centralized database client establishes a new session for every performed transaction, and in opposite to this, middleware behaves as a connection poll for clients. It causes that in middleware new connections for every transaction is not required since the sets of transactions are submitted to the database in parallel using only one session or earlier arranged pool of sessions. At the loads higher than 80 tps the centralized system is becoming saturated, whilst the replicated system still can handle loads within an acceptable performance.

**Scalability summary**

The obtained results show that Theta method provides high level of scalability since for more than 2 replicas system behaves similarly to the configuration with only two database copies. Moreover, when the distant between remote replicas is higher, the response time are expected to grow slightly since there is not much communication required between remote locations – this test are not yet performed because of the environment and research time limitations.

Because of the requirement of the similar efficiency of the machines used as replication nodes (tests are realized on separate physical machines) research tests are performed with up to 10 replicas. However, even though the tests are performed in the worst case scenario with 100% of modification transactions, the overall Theta replication system efficiency with loads higher than 50 tps is much better than in the system with centralized database.

Performed tests results show that the system is efficient with a growing amount of replicas and along with increasing loads it behaves better than a single instance of centralized database. Both these features of the method confirms that the proposed Theta replication method is feasible and suitable for transactional systems with large numbers of distributed nodes.
4.3 Results

4.3.2 Percentage of reads

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10 000 [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>5-100 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0, 20, 40, 60, 80, 99 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>20 [%]</td>
</tr>
</tbody>
</table>

Table 4.6: Scenario with different percentages of reads

According to the scalability experiment results the replication system with more than two replicas built on the basis of the implementation of Theta approach behaves similarly to the system with 2 replicas.

Figure 4.5: Queries percentage influence on response times
Therefore, to avoid the analysis becomes illegible, the additional assumption is made that all the following test scenarios are performed within the system with 2 replicas.

The percentage of queries is an important factor when replication technique is evaluated. This parameter characterizes the behavior of the approach when the application needs to process many queries, many update transactions or both of them. Thus, the percentage of read operations is examined in the scenario presented in table 4.6, where ratio of read to write transactions varies among the subsequent runs. The gained results are presented in the following part of the section.

<table>
<thead>
<tr>
<th>Load [tps]</th>
<th>0%</th>
<th>20%</th>
<th>40%</th>
<th>60%</th>
<th>80%</th>
<th>99%</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>57.0</td>
<td>52.4</td>
<td>50.0</td>
<td>45.8</td>
<td>43.2</td>
<td>39.2</td>
</tr>
<tr>
<td>10</td>
<td>61.0</td>
<td>57.0</td>
<td>53.4</td>
<td>48.0</td>
<td>44.0</td>
<td>39.6</td>
</tr>
<tr>
<td>20</td>
<td>64.4</td>
<td>60.6</td>
<td>53.6</td>
<td>50.0</td>
<td>47.4</td>
<td>40.8</td>
</tr>
<tr>
<td>30</td>
<td>66.0</td>
<td>63.0</td>
<td>58.4</td>
<td>54.4</td>
<td>47.8</td>
<td>41.6</td>
</tr>
<tr>
<td>40</td>
<td>71.2</td>
<td>69.0</td>
<td>66.4</td>
<td>55.6</td>
<td>49.2</td>
<td>42.2</td>
</tr>
<tr>
<td>50</td>
<td>79.2</td>
<td>77.2</td>
<td>68.8</td>
<td>59.2</td>
<td>50.2</td>
<td>41.8</td>
</tr>
<tr>
<td>60</td>
<td>87.6</td>
<td>83.8</td>
<td>75.2</td>
<td>66.8</td>
<td>52.4</td>
<td>43.2</td>
</tr>
<tr>
<td>70</td>
<td>93.8</td>
<td>94.6</td>
<td>81.0</td>
<td>68.6</td>
<td>54.6</td>
<td>43.4</td>
</tr>
<tr>
<td>80</td>
<td>98.8</td>
<td>99.4</td>
<td>87.4</td>
<td>75.0</td>
<td>58.6</td>
<td>45.6</td>
</tr>
<tr>
<td>90</td>
<td>106.6</td>
<td>104.2</td>
<td>94.2</td>
<td>82.8</td>
<td>76.8</td>
<td>57.8</td>
</tr>
<tr>
<td>100</td>
<td>116.6</td>
<td>112.2</td>
<td>101.8</td>
<td>103.0</td>
<td>95.6</td>
<td>67.6</td>
</tr>
</tbody>
</table>

Table 4.7: Test results for various percentages of queries

Similarly to the previous scenarios, in the following scenario the total number of clients changes starting from 5 up to 100. The amount of read transactions is set to 0, 20, 40, 60, 80 and 99% of all submitted transactions. This percentage is not set to 100 because a read-only system would be implemented in a completely different way than a replication system. Number of conflicting transaction is again set to 20%.

Test results presented in fig. 4.5 and in respective table 4.7 show the influence of varying percentages of queries on the mean response times in a system with Theta approach implemented. The response times for Theta replication goes down while the query percentage increases. The reason for such system behavior
4.3 Results

is the decrease in a number of transaction interferences since query does not interfere with other queries.

It is obvious that when read-only transactions are scattered over many replicas the average load in a single replica is lowered as it is balanced among all replicas, therefore the average response times are shorter [3, 54, 61, 76]. The more replicas in the system, the better it behaves according to the read-only transactions. Since Theta approach enables data reading from local replicas, mean response time for the approach is getting shorter when the percentage of queries increases.

4.3.3 Conflicting transactions ratio

Similarly to the percentage of queries described earlier, the percentage of conflicting transactions is the replication parameter that significantly influences the replication system behavior.

![Percentage of conflicting transactions](image)

Figure 4.6: Conflicts ratio influence on the approach response times
4.3 Results

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10 000 [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>5-100 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>10, 20, 30, 40 [%]</td>
</tr>
</tbody>
</table>

Table 4.8: Scenario with different conflicting transactions ratios

The ratio of conflicting to non-conflicting transactions in a set of all submitted transactions considerably depends on the kind of the application. However, in the majority of database systems used for real world applications, the range of this parameter floats around 10% to 30% of conflicting transactions among all submitted transactions. Therefore, 10, 20, 30 and 40 ratios are defined for the purpose of this experiment. Such values of conflicting transactions ratio allows to examine the approach usefulness for various types of applications. Scenario definition is described in details in table 4.8.

<table>
<thead>
<tr>
<th>Load [tps]</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>57.2</td>
<td>57.2</td>
<td>59.6</td>
<td>58.2</td>
</tr>
<tr>
<td>10</td>
<td>67.0</td>
<td>63.4</td>
<td>65.8</td>
<td>62.6</td>
</tr>
<tr>
<td>20</td>
<td>64.6</td>
<td>64.8</td>
<td>68.0</td>
<td>66.0</td>
</tr>
<tr>
<td>30</td>
<td>69.4</td>
<td>67.2</td>
<td>70.0</td>
<td>69.4</td>
</tr>
<tr>
<td>40</td>
<td>79.4</td>
<td>77.0</td>
<td>78.8</td>
<td>80.2</td>
</tr>
<tr>
<td>50</td>
<td>88.8</td>
<td>86.8</td>
<td>88.6</td>
<td>91.6</td>
</tr>
<tr>
<td>60</td>
<td>99.4</td>
<td>98.8</td>
<td>99.6</td>
<td>103.2</td>
</tr>
<tr>
<td>70</td>
<td>110.6</td>
<td>106.6</td>
<td>110.2</td>
<td>113.6</td>
</tr>
<tr>
<td>80</td>
<td>122.4</td>
<td>118.2</td>
<td>121.8</td>
<td>123.4</td>
</tr>
<tr>
<td>90</td>
<td>131.6</td>
<td>129.0</td>
<td>132.0</td>
<td>134.6</td>
</tr>
<tr>
<td>100</td>
<td>141.2</td>
<td>136.8</td>
<td>139.0</td>
<td>139.0</td>
</tr>
</tbody>
</table>

Table 4.9: Test results for various conflicting transactions ratios

Test results presented in fig. 4.6 and in respective table 4.9 show the way that mean response times for Theta method are affected by submitting transaction with various percentages of conflicting transactions. The mean response times
measured for the method are going down as the number of conflicting transactions increases.

The reason of response time decrease is the fact that the mutual interference of transactions is higher when more conflict in middleware appears and need to be prevented. For long lasting transactions this decrease would be higher. However, the method is designed for OLTP systems processing mainly transient transactions, therefore, long transactions are not considered in this scenario.

The presented experiments results show that the method is suitable for the systems with various number of conflicting transactions. When the number of clients grows the average response times increase as well, however this increase is still within an acceptable range.

### 4.3.4 Size of database copies

To evaluate the size of the database copies influence on the transaction processing with the usage of Theta replication technique, experiments with different sizes of replicas are conducted. Again, the load in experimental scenarios changes from 5 up to 100 tps and 100% of submitted transactions are updates.

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10K, 100K, 1M, 10M, 100M [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>5-100 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>20 [%]</td>
</tr>
</tbody>
</table>

Table 4.10: Scenario with table indexes – transient transactions

The tests are performed for varying sizes of database tables with indexes, which simulates transient transactions (scenario in table 4.10), and without key indexes – long lasting transactions (table 4.12). Test results are presented respectively in tables 4.11 and 4.13.

Removal of table indexes in the scenario without table indexes causes that full table scans need to be performed on these tables while processing transactions.
4.3 Results

This slightly increases the duration of their processing in the database, and in a consequence queries run against non-indexed tables appropriately simulates long lasting queries. Therefore, results gained in these 2 scenarios show how the system is affected when the type of the load changes.

In the scenarios where table indexes exists in database tables, the size of tables in each database copy changes from 10 thousand rows up to 100 million rows. Since database response times for scenarios without indexes increased considerably, tests for this scenario are performed only up to 1 million rows.

![Different size of tables](image)

**Figure 4.7**: Database size influence on mean response times – tables with indexes

*Mean response times* for replicated system with key indexes existing are presented in fig. 4.7. It is noticeable that *response times* grow slightly for the lower loads. For the loads higher than 50 tps the *average database response time* grows around 10 times (from 3 ms for 10K rows to 29 ms for 100M rows) and *mean response time* increases faster. However, this growth is fairly slow with a reference
4.3 Results

to the growth of the database response time, and the system continuously replies within acceptable limits.

<table>
<thead>
<tr>
<th>Load [tps]</th>
<th>10K</th>
<th>100K</th>
<th>1000K</th>
<th>10M</th>
<th>100M</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>57.0</td>
<td>58.0</td>
<td>58.0</td>
<td>57.2</td>
<td>60.2</td>
</tr>
<tr>
<td>10</td>
<td>61.0</td>
<td>63.2</td>
<td>60.6</td>
<td>61.0</td>
<td>67.6</td>
</tr>
<tr>
<td>20</td>
<td>64.4</td>
<td>65.8</td>
<td>63.2</td>
<td>65.2</td>
<td>69.4</td>
</tr>
<tr>
<td>30</td>
<td>66.0</td>
<td>67.4</td>
<td>67.8</td>
<td>67.8</td>
<td>71.4</td>
</tr>
<tr>
<td>40</td>
<td>71.2</td>
<td>74.0</td>
<td>73.2</td>
<td>74.8</td>
<td>78.8</td>
</tr>
<tr>
<td>50</td>
<td>79.2</td>
<td>81.6</td>
<td>80.0</td>
<td>85.8</td>
<td>95.2</td>
</tr>
<tr>
<td>60</td>
<td>87.6</td>
<td>87.0</td>
<td>86.8</td>
<td>92.6</td>
<td>99.6</td>
</tr>
<tr>
<td>70</td>
<td>93.8</td>
<td>93.6</td>
<td>94.6</td>
<td>109.6</td>
<td>111.4</td>
</tr>
<tr>
<td>80</td>
<td>98.8</td>
<td>101.0</td>
<td>106.4</td>
<td>113.8</td>
<td>119.4</td>
</tr>
<tr>
<td>90</td>
<td>106.6</td>
<td>119.6</td>
<td>118.0</td>
<td>119.2</td>
<td>127.2</td>
</tr>
<tr>
<td>100</td>
<td>116.6</td>
<td>119.6</td>
<td>121.8</td>
<td>128.8</td>
<td>137.2</td>
</tr>
</tbody>
</table>

Table 4.11: Test results for transient transactions

Scenario without table indexes, which simulates long lasting transactions, is presented in table 4.12. Mean response times for this scenario are graphically presented in fig. 4.8, while tests results are supplied in table 4.13.

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data management</td>
<td>Theta replication method</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
</tr>
<tr>
<td>Size of tables</td>
<td>10K, 100K, 1M [rows]</td>
</tr>
<tr>
<td>Load</td>
<td>5-100 [tps]</td>
</tr>
<tr>
<td>Queries percentage</td>
<td>0 [%]</td>
</tr>
<tr>
<td>Conflicts ratio</td>
<td>20 [%]</td>
</tr>
</tbody>
</table>

Table 4.12: Scenario without table indexes – long lasting transaction

In the scenario when there are no key indexes in tables, the mean database response time grows rapidly and for 1 million rows (mean database response time is 59 ms) this time is more than 10 times longer than for the non-indexed tables.

Such a high growth in the time of a single long transaction processing in database causes that system performs efficiently only for the lower loads. For the
higher loads in connection with tables containing 1 million rows \textit{mean response time} grows very quickly, which may cause serious bottlenecks in a system.

The experiment results presented in the scenarios in this section prove that Theta method allows to gain very good \textit{mean response times} for large amount of short transactions.

On the other hand, when the database response time grows significantly, the approach is not suitable when it comes to the higher loads. However, the method can still be applicable for synchronization of such database copies when the load is lower.

![Different size of tables](image)

Figure 4.8: Database size influence on mean response times – long lasting transactions for tables without indexes

Since the approach is destined to transactional, multi-node systems, issue related to high loads for long lasting transactions is not further analyzed.
4.4 Summary

<table>
<thead>
<tr>
<th>Load [tps]</th>
<th>10K</th>
<th>100K</th>
<th>1000K</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>57.0</td>
<td>76.8</td>
<td>88.2</td>
</tr>
<tr>
<td>10</td>
<td>61.0</td>
<td>78.4</td>
<td>93.6</td>
</tr>
<tr>
<td>20</td>
<td>64.4</td>
<td>86.8</td>
<td>98.4</td>
</tr>
<tr>
<td>30</td>
<td>66.0</td>
<td>100.4</td>
<td>147.2</td>
</tr>
<tr>
<td>40</td>
<td>71.2</td>
<td>128.2</td>
<td>222.8</td>
</tr>
<tr>
<td>50</td>
<td>79.2</td>
<td>156.6</td>
<td>344.6</td>
</tr>
<tr>
<td>60</td>
<td>87.6</td>
<td>184.6</td>
<td>510.8</td>
</tr>
<tr>
<td>70</td>
<td>93.8</td>
<td>262.8</td>
<td>709.4</td>
</tr>
<tr>
<td>80</td>
<td>98.8</td>
<td>249.0</td>
<td>920.6</td>
</tr>
<tr>
<td>90</td>
<td>106.6</td>
<td>299.0</td>
<td>1130.8</td>
</tr>
<tr>
<td>100</td>
<td>116.6</td>
<td>350.0</td>
<td>1346.2</td>
</tr>
</tbody>
</table>

Table 4.13: Test results for long lasting transactions

4.3.5 Portability

To perform portability tests of Theta replication approach, replication environment is prepared in configuration with several hardware and software platforms. On the purpose of this experiment Theta method is implemented for the SUN SUN SPARC, AMD and Intel platforms, SUN Solaris and RedHat operating systems and Oracle and PostgreSQL database engines.

The performed tests in various configurations confirm Theta replication method portability and possibility of the usage in heterogeneous environments based on different operating systems, various hardware platforms and RDBMS.

4.4 Summary

Similarly to the evaluation methods of the replication approaches available in literature, Theta replication method evaluation is performed in a dedicated test environment, which allows to verify whether it is applicable for the certain solutions that it is designed and implemented for.

The chapter covers the issues related to the experimental evaluation of new Theta replication method in aspects of its scalability and portability. Presented tests results and their analysis show that Theta replication provides scalable and
non-conflicting mechanism that allows on keeping multiple replicas synchronized. The method ensures high scalability and at the same time provides acceptable efficiency of data replication process. Theta method ensures complete data consistency and its performance is very good comparing to a centralized system for all testing workloads.

- Performed experiments show that the system still responds efficiently while the amount of replicas is growing and for higher transaction loads replicated system behaves better than a single instance of centralized database.

- In-laboratory method implementation of the system with Theta replication is compatible with the most of existing database systems. Since the approach is based on the execution of the stored procedures in remote replicas, it can be used in heterogeneous environments with no difficulty.

- Conducted experiments confirm Theta replication method portability and possibility of the usage in environments based on different operating systems, various hardware platforms and RDBMS.
Chapter 5

Real life evaluation

Table 2.2 (presented in 2.6) provides the analysis of the data replication approaches in the aspects of theirs possible usage in various data management systems. Majority of the researches related to data replication in multi-agent systems (which in fact are representative for distributed multi-node systems) focuses on ensuring appropriate fault tolerance of the system or addresses security issues. Therefore, Theta replication method designed for distributed transactional systems is chosen for replication based information sharing in real life multi-agent distributed IBIS system [65, 90].

In the following chapter there is an introduction to the IBIS platform and then there is an extensive presentation of the practical implementation of Theta replication method in IBIS system. Afterwards evaluation scenarios are presented, which includes verification of failure resistance and detailed information about the performed tests of IBIS functionalities in replicated environment.

5.1 Introduction to IBIS system

The main purpose of the IBIS system is to support a user in detection of possible illegal activities in the Internet. IBIS is the system that continuously monitors the Internet and classifies objects found (WWW pages first of all) according to the similarity of the content of those objects to the user’s defined profiles describing the area of interests. IBIS platform is a multi-agent and multi-user system. Its implementation consists of a set of machines on which particular agents run.
Therefore, the system needs to continuously gather information from the Internet and then to perform the analysis to find activities that can be illegal or can be arrangements of such activities. The IBIS system is supposed to fulfill the following demands:

- limitation of manual work during investigation,
- repeatable usage of developed procedures by independently working operators,
- minimization of repeatable actions.

### 5.1.1 System logical structure and architecture

In fig. 5.1 there is a presentation of the logical structure of the IBIS system [90]. System consists of the three main subsystems working directly on the data stored in the system:

- data gathering subsystem, which gathers information from the Internet, evaluate this data and stores it in the database,
- alert subsystem, which generates alerts and notifications,
- interface subsystem, which enables users to access data.

Both alert and interface subsystems presents to users data collected in the database with the use of models, samples and profiles which also are stored in the database. The difference is that interface subsystems reacts on the users’ questions, while alert subsystems informs a user when interesting event takes place in the system.

The IBIS subsystems apply configuration parameters stored in the database. Because the system is based on the agents accessing data in distributed environment, these parameters need to be up to date in every subsystem and for every autonomous agent.
5.1 Introduction to IBIS system

**Principle of operation**

System IBIS is going to process items related to the particular Internet location. This processing is comparison of items with profiles defining features of content that are looked for in the system. The processing of a single item includes several phases realized by autonomous agents in fixed order, but not necessary in direct time sequence. After finishing a task, an agent picks up another task from a shared pool of tasks and starts its processing.

Each phase of processing is realized by a certain type of agent, which performs necessary processing and is prepared to process specific contents. At the beginning usually no data is related to the processed item. However, during processing of item, data is collected in a database as agents gather or calculate it. This causes that although agents do not exchange data directly, state of processing of the item changes, and data achieves the required final form.

When an agent needs to pick up a new task to process, connects to the agent distributing tasks. A task is selected by the other agent on the basis of its strategy.
and other factors and rules, which can be parameters from the database or from outside the system given by an application administrator.

**IBIS architecture**

IBIS system designed for network environment consists of two components. The first one is a set of agents which process tasks. Agents are classified depending on kind of that they process. Every agent uses the same mechanism to receive tasks, and requires specified information like from where to get task, where there are needed data or where to put results of processing. The other component is a set of agents responsible for maintenance of the whole system and interaction with users.

![IBIS system architecture](Source: [90])

**Figure 5.2: IBIS system architecture (Source: [90])**

In fig. 5.2 there is a demonstration of the architecture of the system in the context of data storing. Every cluster consists of a certain amount of nodes, where there is only one master-agent and a number of specialized processing agents (number of these agents can change whilst processing).
5.1 Introduction to IBIS system

IBIS platform

IBIS platform, which in fact is an agent environment, is presented in fig. 5.3. Physically it consists of a set of computers, on which processes of agents are run. Applying such an architecture the adequate computing power is ensured, while it is also possible to easily extend power of the system by adding new computers.

IBIS platform consists of two agent types – Processing Agents and Special Agents. Processing Agent which performs specific tasks supplied by users. There are many types of Processing Agents in IBIS platform, and each of them performs particular tasks. Special Agents required to ensure appropriate behavior of the system. There are three types of Special Agents:

Figure 5.3: Architecture of IBIS platform (Source: [65])
5.1 Introduction to IBIS system

- **Node Supervisor Agent** which controls and manages works performed by the platform. This process is realized by management of the amount and type of Processing Agents that are run.

- **Cluster Supervisor Agent** is an agent that is run in every processing node, and invokes and controls Processing Agents.

- **Dispatcher Agent** divides tasks which are to be processed by the platform among Processing Agents.

On the purpose of the research the following three types of Working Agents are considered:

- **SeedsAgent** is a crawler agent. It generates Objects addresses according to strategy defined in a starting address. Typically, if URL address represents input for SeedsAgent, agent generates new URLs accessed from the starting URL. In IBIS system these generated addresses are used later by other Working Agents, for instance to download content.

- **HtmlParserAgent** downloads from WWW network documents located at specified URLs. Then HtmlParserAgent agent analyzes downloaded documents and selects required text. Html is parsed with the usage of HTML Parser 1.6 [75], which is a Java library used to parse HTML in either a linear or nested fashion. Finally, HtmlParserAgent writes results of the performed work to database.

- **SearchWVAgent** supplies possibility of searching whole database or its part to find required data. SearchWVAgent uses Apache - Tika 0.3 library [112] to analyze content, while searches for text patterns.

Moreover, IBIS platform uses two data repositories – Agents Repository and Database. **Agents Repository** is used to store binary codes of Processing Agents and make them available in remote locations. **Relational Database** is used to store data of the IBIS platform. To optimize the functionality of the platform, the database is divided into three separate databases: Strategy database, Configuration database and Working database.
5.1.2 Agents interactions

The interactions between particular types of agents in IBIS system is presented in figure 5.4.

To coordinate work of every node in IBIS system Cluster Supervisor Agent controls work of every node by:

- deciding which type of Processing Agent is run by Node Supervisor Agent,
- defining amount of running Processing Agents.

Cluster Supervisor Agent realizes this process by sending profiles containing the above information to Node Supervisor Agent. Node Supervisor Agent periodically informs Cluster Supervisor Agent about the node load, which is also...
a confirmation of the stable and correct work of this node. Cluster Supervisor Agent and Node Supervisor Agent interaction is based on the realization of the following tasks:

- After receiving information from Cluster Supervisor Agent, Node Supervisor Agent invokes appropriate Processing Agents,

- Node Supervisor Agent submits to Processing Agents data necessary for their work,

- Node Supervisor Agent controls Processing Agents that are run on behalf of particular Node Supervisor Agent,

- Processing Agents periodically reply to theirs Node Supervisor Agent sending reports from the work already performed by them.

After being run Processing Agents register at Dispatcher Process to gain the work they are supposed to perform. Dispatcher verifies the type of Processing Agents and assigns them appropriate tasks. After finishing work Processing Agent informs Dispatcher.

5.2 Database architecture

Relational database is used to store data in the IBIS platform. To optimize platform functionality database is divided into three separate databases:

- Configuration database,

- Strategy database,

- Working database.

Most of the communication with database is realized locally and is managed by database module. Replicated data is accessible by the database module supported by data replication module. Database module in IBIS system ensure communication with database. The main goals of this module are:
5.2 Database architecture

- physical storage for data related to multiple database,
- mapping of logical addresses accessed by outgoing application interfaces to physical data location in databases,
- logical translation between relational and object data models,
- data security, which means ensuring data access control mechanisms and transactional management.

Database structure in IBIS system is presented in fig. 5.5. The following subsections contain details on particular databases.

![Figure 5.5: IBIS databases (Source: [65])](image)

**IBIS databases**

**Strategy database** consists of strategies that are distributed by Cluster Supervisor Agent to Node Supervisor Agents. The stored strategies include information
about the type of agents that are required to be run and the ticket to the agent repository which allows to gain and use the code of agent. Strategy database is shown in fig. 5.6.

Figure 5.6: Strategy database
5.2 Database architecture

**Working database** is the database that stores actual data which are processed by agents. This database is accessed by agents while they normally work on processing tasks. Working database is presented in fig. 5.7.

![Working database diagram](image)

**Figure 5.7: Working database**

**Configuration database** consists of a list of any resources existing in the system, called web objects, including their status and work done for them. This list also contains an information about tasks grouping and other information used
mainly to improve the quality of the users’ operational work in the system. Fig. 5.8 presents schema of Configuration database.

Figure 5.8: Configuration database
5.3 Replication implementation for IBIS

Data used by Dispatcher is also stored in Configuration Database. Since Dispatcher controls tasks processing, it has to modify appropriate data in Configuration Database. This process includes the following two operations:

- update the current state of task,
- prepare the set of tasks that are to be done as a single group of tasks.

Data in the Configuration Database includes all tasks which are ready to process and which are distributed by Dispatcher.

5.3 Replication implementation for IBIS

While performing tasks on the purpose of the common goal, agents have to work on consistent data, easily and efficiently accessed from the servers on which processes of agents run.

In general data replication for IBIS system addresses the following issues:

1. Agents in IBIS system need continuously access common steering parameters which controls their behavior in a dynamically changed environment, therefore it is required to replicate steering parameters.

2. Jobs distribution among agents is performed by Dispatcher agent. This process also needs to be supported by data replication. Distribution of jobs strongly depends on the data related to the current assignment of tasks to particular agents, as well as on the current priorities of the jobs (since tasks priorities are continuously adjusted). If data is stored in one location only, one central Dispatcher manage all of the tasks which causes bottlenecks in the overall process. Therefore, independent Dispatchers agents need to be run in every processing location. Shared by the Dispatchers information about particular tasks assignments and priorities must be coherent and available by every Dispatcher with restricted time limitations, thus data replication is required to fulfill this demand.
5.3 Replication implementation for IBIS

5.3.1 Requirements

Fig. 5.2 presents the architecture of the system in the context of data storing. Every cluster of the system consists of a certain amount of nodes, where there is only one master-agent and a number of specialized processing agents. All the agents in any cluster need to use shared data, therefore it is necessary that data is available from every agent and data is coherent and up to date in each cluster.

Databases in every cluster of the system cooperate with one other ensuring full data replication or for some data a partial replication. Thanks to the replication control parameters in any cluster contain the same information, which ensures effective management of the work of any agent in the system. Similarly, the users’ queries stored in database, search patterns and profiles are locally available, which makes their work easier, no matter of their location with reference to the cluster (it can be the nearest cluster, the least loaded one or any other).

On the contrary, it is a high volume of data that is gathered from the Internet. This data is processed which also gives a lot of data as a results of this processing. Moreover, this data does not have to be accessible in every node all the time, thus it can be stored locally or partially replicated to achieve better performance. However, partial replication implementation issues for IBIS system are out of the scope of this research.

In consequence the following requirements are defined and need to be satisfied by the implementation of Theta replication method to be successfully adapted in IBIS system:

- run and manage more remote agents at once,
- coherent and up to date data accessible in every location of replication,
- resistance to failures,
- easy adaptation and implementation of the approach for already working IBIS system.

Running and managing more Cluster Supervisor Agents (generates one transaction per second) at allows to perform jobs faster as many Working Agents can be invoked per every single Cluster Supervisor Agent. The usage of centralized
database in the implementation of the IBIS system enforces the usage of the only one central Cluster Supervisor Agent running in the whole system. This leads to the bottlenecks in the centralized database. Additional remote replicas with coherent and up to date should allow to run multiple Cluster Supervisor Agents without downgrading performance of the whole system. This may significantly increase processing efficiency in IBIS system and in fact increases scalability of the IBIS system.

5.3.2 Data replication architecture

The following section introduces data management in IBIS including presentation of data replication architecture, middleware inter-process communication on the purpose of replication realization and the issues related to the data access and data flows in the replicated system.

The main purpose of applying data replication in IBIS system is sharing common strategies among IBIS agents and allowing to run multiple copies of Dispatcher processes. Two replicated database instances are allocated for Strategy and Configuration databases. This databases are used for storing common steering parameters, application control parameters, as well as for data related with tasks managed by Dispatchers. All this data is replicated among every remote location. Schema presented in fig. 5.9 introduces the architecture of the IBIS platform with data replication implementation.

Theta replication method is used to replicate data in the IBIS system. This particular way of data replication realization allows to preserve previously implemented functionalities, while on the other hand allows to easy adapt Theta method to the requirements of the data replication in IBIS, without the necessity of performing changes in the core of the implemented system.

Middleware replication subsystem for Configuration database is connected with every instance of Dispatcher Agent, which means that every Dispatcher Agent has an associated replica (consisting of data copy related to the task management) that it works on. Similarly, every Cluster Supervisor Agent is associated with appropriate Strategy database replica.
To supply a work to a particular Processing Agent, Dispatcher Agent requests task identifier from local replica, updates status of this task as attached in this replica and submits work to this Processing Agent. Details of the attached work Processing Agent receives from a centralized Working database. Processing Agent also communicates centralized database to keep temporary data required in processing as well as for permanent storing of processing results.

Steering parameters are replicated to every site of IBIS system allowing to run Cluster Supervisor Agents instances in each site of the system. Dispatcher agents running on many servers (sites) cooperate with theirs associated subsets of Processing agents, usually within the same server machines. Dispatcher and Clus-
5.3 Replication implementation for IBIS

ter Supervisor agents work with data replicas through a middleware components connecting directly to CMAN process it works with.

![Diagram of IBIS agents and processes communication](image)

Figure 5.10: IBIS agents and processes communication

Details on communication between particular middleware processes realizing data replication, IBIS agents communication, and finally interactions between IBIS agents and middleware replication subsystem is presented in fig. 5.10.
5.3 Replication implementation for IBIS

5.3.3 Data flow and data management in replicated IBIS

Activity diagram representing data flow for IBIS system with Theta replication implementation is presented in fig. 5.11.

Diagram does not include detailed information for middleware interaction while conflict resolution is performed. Middleware data flow in the diagram is represented by Middleware Replication Module which covers conflict resolution issues, and is widely described in chapter 3.

![Activity diagram for IBIS replication](image)

Figure 5.11: Activity diagram for IBIS replication

Working Agents operates on huge amounts of data. However, this data is related only to connected Working Agent and has no impact on other agents work, therefore Working Database is built as central database.

To enable access to data stored in Working database of IBIS system IBATIS framework [40] is used. IBATIS provides a mechanism that allows on mapping data stored in relational database to object model. IBATIS framework cooperates
5.3 Replication implementation for IBIS

with Spring framework [100] which is used to ensure transactions for database operations.

![Diagram of Communication in IBIS with replication]

Figure 5.12: Communication in IBIS with replication

Direct data access is based on the usage of Data Access Object (DAO), which ensures appropriate set of functions for data management. DAO in IBIS system is responsible for:

- preparation of SQL queries,
5.3 Replication implementation for IBIS

- communication with database,
- transactions management,
- query parameters mapping,
- mapping of gained result sets.

DAO objects are used by Services. In system only Services are able to access data directly in database on behalf of particular component of IBIS. This allows to separate data access management from business logic.

General idea of communication in IBIS system with replication implementa-
tion is presented in fig. 5.12. Middleware interfaces for external communication are in details presented in the section 4.1 in fig. 4.2.

To communicate with the appropriate database copy, Dispatcher Agents and Cluster Supervisor Agents connect with Connection Managers in relevant middleware located in the same cluster as related agents. Both Dispatcher Agents and Cluster Supervisor Agents communicate with CMAN process using connection handler based on TCP socket communication[102]. Connection handler implemented for Theta approach provides a mechanism that allows to use object oriented technology for implementation of data replication in IBIS system. Dispatcher Agent submits transaction to CMAN, where submitted transaction is processed and afterwards is submitted to the appropriate replica.

Processing Agent requests Dispatcher Agent for work by sending information about its type and kind of operation it can perform. Dispatcher calls replica throughout middleware to get requested task, then sets the state of the task as assigned, and finally submits the task to the Processing Agent.

Dispatcher periodically scans database with tasks. After finding a work, for which time limit is exceeded, Dispatcher changes its status to Ready to process. While scanning database with tasks Dispatcher Agent removes tasks that are already finished as well as adds new tasks subsequent to the finished tasks on the basis of current system strategy.
5.3 Replication implementation for IBIS

5.3.4 Implementation details

To ensure compatibility with the IBIS system based on Java platform, Theta replication implementation for IBIS Internet monitoring system is realized in Java programming language.

![Deployment diagram for Theta replication management in IBIS](image)

Figure 5.13: Deployment diagram for Theta replication management in IBIS

Communication between the middleware components for IBIS system data replication implementation is based on the usage of Remote Method Invocation (RMI) [39, 71]. Fig. 5.13 presents the deployment diagram of the Internet monitoring system with the implementation of data replication based on Theta method. Each middleware server consists of its own RMI registry which stores
5.3 Replication implementation for IBIS

stubs for remote methods of particular processes residing on this server. This methods are used for information exchanging between processes. Information is submitted by a source process and then remote method of destination process applies the message into this process input queue.

Figure 5.14: Data flow for iBATIS data mapper (Source: [40])

The communication between connection manager and global identifier generator is based on TCP sockets [102] usage. In the implementation for IBIS system there is only one GIDG process dedicated for servers in every site which runs within its own Java Virtual Machine (JAVA VM) and is located in the same server machine where the other middleware processes run. However, it is possible to exclude GIDG from the common server, and invoke within Java VM that runs on the dedicated server.

Introduced earlier fig. 5.12 presents the way that agents in IBIS system communicate with centralized Working Database and with replicas of distributed
5.4 Evaluation scenarios and results

The performance of the management of steering parameters and strategies is crucial for the overall IBIS system performance. This performance grows while the number of Cluster Supervisor Agents and related to them Processing Agents is increasing. Since scalability tests are performed and presented in details in 4.2, such evaluation of Theta replication implementation in IBIS system is performed only for the very important process of strategies and parameters management.

Validation of data integrity in every replicas in the whole replicated IBIS system and verification wether replicas contain the latest, up to date information (strategies, steering parameters, task management data, etc.) is analyzed during IBIS functionality tests. These tests are concentrated on verification of the replication process correctness during realization of the main activities realized in the system (web interface management, jobs and task assigning, patterns searching, strategy management).

Evaluation of the Theta method fault tolerance and implementation requirements according to the easiness of its adaptation to real system demands are also verified and presented.

Experiments exploring system properties as well as analysis of particular system use cases are conducted on several, concurrently working machines. A series of Working Agents are run in every machine to realize particular tasks. Each machine running Working Agents can be treated as an autonomous site and is connected to a separate configuration database (replica). Implementation issues
related with data replication in IBIS is presented in the section 5.3.4, while detailed specification of the test environment, software and hardware used for IBIS system implementation can be found in the appendixes B and C.

The following sections provides evaluation scenarios prepared for functional tests in replicated IBIS system. First there is a presentation of issues related to the exploration of failure resistance of Theta replication method in IBIS with replication. Afterward, there are another scenarios in which the correctness of IBIS main functionalities is examined in the system based on Theta replication implementation.

5.4.1 Failure resistance

Resistance to failures of replicated components is required in most of system used in practice, and it is also required in replicated IBIS system. Thus, Theta replication method used on the purpose of IBIS system is verified against its resistance to failures. Test include verification of system behavior in cases of failures of particular replicas and middleware components. Fault tolerance issues for Theta replication method are presented in the section 3.3.5, while the following section presents the description and results gained for failure scenarios in IBIS system with replication.

Aim of test

The aim of this test is verification of middleware transaction recovery process in IBIS system with data replication implementation in a worst case, which is middleware and database failure.

Test schema

Schema of the test is as follows:

1. Enable middleware transaction logging.
2. Create database on-line backup.
3. Simulate failure of middleware and database.

4. Restore database datafiles.

5. Recover database.

6. Apply transactions from middleware transaction log.

7. Start Connection Manager and handle client connections.

**Test details**

*Ad 1. Enable middleware transaction logging.*

If transaction logging in middleware is not active, it should be enabled in replication configuration file by setting archive_enabled parameter to true:

```plaintext
archive_enabled=true
```

WAL archiving for PostgreSQL database should also be enabled. If it is not, WAL archiving should be set up. The command which will copy archivable WAL segments to the directory /data/archivedir is:

The shell command to use, that is specified by the archive_command configuration parameter, should be placed in the postgresql.conf file.

```plaintext
archive_command = 'cp -i %p /data/archivedir/%f < /dev/null'
```

*Ad 2. Create database on-line backup.*

To create database on-line backup the following steps should be done:

1. As a superuser, issue the command:
2. Perform the backup of data, using any convenient file-system-backup tool such as tar or cpio.
   It is neither necessary nor desirable to stop normal operation of the database while doing backup.

3. As a superuser, issue the command:

   ```
   SELECT pg_start_backup('db_online_bck');
   
   # tar cvf /backup/pg_db.tar /postgres/8.4/data
   ```

   **Ad 3. Simulate failure of middleware and database.**

   Defect in middleware and database is simulated by turning off power in middleware machine containing both middleware replication software and replica.

   **Ad 4. Restore database datafiles.**

   The procedure of restoring datafiles is as follows:

   1. Stop the postmaster, if it’s running.

      ```
      # kill -INT `head -1 /usr/local/pgsql/data/postmaster.pid`
      ```

   2. Copy the contents of the pg_xlog subdirectory of the data directory, as it may contain logs which are not archived before the system went down.
5.4 Evaluation scenarios and results

3. Clean out all existing files and subdirectories under the data directory and under the root directories of any tablespaces used.

4. Restore the database files from backup dump. Set the datafiles ownership to the database system user and set the right permissions. To restore datafiles from tar backup issue the following commands:

```
# cd /postgres/8.4/data
# tar xvf /backup/pg_db.tar
# chown -R postgres:postgres /postgres/8.4/data
```

**Ad 5. Database point-in-time recovery (PITR).**

The following steps are required to perform database point-in-time recovery:

1. Any files present in pg_xlog/ directory should be removed.

2. Subdirectory pg_xlog/archive_status/ must exist.

3. If there are unarchived WAL segment files (not archived logs saved previously), they should be copied into pg_xlog/.

4. Create a recovery command file recovery.conf in the data directory. The restore_command must be specified in recovery.conf, which tells PostgreSQL how to get back archived WAL file segments. Like the archive_command, this is a shell command string. It may contain %f, which is replaced by the name of the desired log file, and %p, which is replaced by the path name to copy the log file to. The shell command to use, that is specified by the restore_command configuration parameter:

```
restore_command = 'cp /data/archivedir/%f %p'
```
5. To prevent ordinary users from connecting until the recovery is finished, pg_hba.conf should be temporarily modified, allowing to connect database from desired IP address only.

6. Start the postmaster which should run into recovery mode and proceed to read through the archived WAL files it needs. Upon completion of the recovery process, the postmaster will rename recovery.conf to recovery.done (to prevent accidentally re-entering recovery mode in case of a crash later) and then commence normal database operations.

7. The contents of the database should be verified to ensure that database is recovered as it should be. If all is well, pg_hba.conf should be restored and database should work properly.

Ad 6. Apply transactions from middleware transaction log.

To finish the database recovery process, transactions that are stored in middleware transaction log should also be applied. To do so, middleware Recovery Manager process RCVMAN should be run. For instance to recover transactions with desired Transaction Identifiers, the following command should be issued:

```
java -cp .;D:\java\lib\log4j-1.2.14.jar;D:\java\lib\postgresql-8.4-701.jdbc3.jar rcvman
RCVMAN> recover from 7800401 to 7809848;
```

Ad 7. Start Connection Manager and handle client connections.

After transaction recovery in middleware is finished, RCVMAN process allows CMAN to start handle connections and transactions are processed in middleware in normal way.

5.4.2 Web Interface

Aim of test
5.4 Evaluation scenarios and results

The aim of this test case is to verify Web Interface behavior in tasks maintaining for Crawling process.

Figure 5.15: Web Interface control panel for IBIS system

Test schema

The test of Web Interface cooperation with replicated Configuration Database are performed with the following schema:

1. Request of creating new task and task instance.
2. Adding new task including new job.
3. Creating task with web object.
4. Creating task with web object type.
5.4 Evaluation scenarios and results

5. Adding task with web object, web object type and related domain.

6. Creating automatic Crawler task.

Test details

Tasks in IBIS system are submitted through Web Interface control panel. An example of Crawler task creating with the usage of Web Interface is presented in fig. 5.15.

While creating Crawler tasks, data is stored in different tables of Configuration database, therefore the following tests are conducted to verify correctness of data writes in appropriate tables.

The following part of the section contains details of performed test scenarios.

Ad 1. Request of creating new task and task instance.

Creating a task in IBIS system is always connected with the necessity of creation task instance which is related to the created task. To verify correctness of adding task and related task instance the following steps are performed:

1. To create new task in IBIS user panel (User Panel / Task ordering / Crawler / Single) the form is filled with the following parameters:
   Start URL = http://awxcnx.us/
   MAX Depth = 2
   Accept URL = CRAWL\_SINGLE\_DOMAIN
   Write URL = CRAWL\_SINGLE\_DOMAIN
   Order Ownership = crawling
   Result Data Set = CRAWLING\_01\_TEST\_TASK\_INST

2. Data verification in Configuration Database is based on confirmation of data existence in appropriate tables in every replica in system. The proper data is confirmed in each replica in tables task and task_instance.

Ad 2. Adding new task including new job.
5.4 Evaluation scenarios and results

1. To create new task in IBIS user panel (User Panel / Task ordering / Crawler / Single) the form is filled with the following parameters:
   - Start URL = http://awxcnx.us/
   - MAX Depth = 1
   - Accept URL = CRAWL_SINGLE_DOMAIN
   - Write URL = CRAWL_SINGLE_DOMAIN
   - Order Ownership = crawling
   - Result Data Set = CRAWLING_02_TEST_JOB

2. Data verification in Configuration Database is based on confirmation of data existence in appropriate tables in every replica in system. The proper data is confirmed in each replica in tables task and active_job.

Ad 3. Creating task with web object.

1. To create new task in IBIS user panel (User Panel / Task ordering / Crawler / Single) the form is filled with the following parameters:
   - Start URL = http://awxcnx.us/web/object/
   - MAX Depth = 2
   - Accept URL = CRAWL_SINGLE_DOMAIN
   - Write URL = CRAWL_SINGLE_DOMAIN
   - Order Ownership = crawling
   - Result Data Set = CRAWLING_03_TEST_WEB_OBJECT

2. Data verification in Configuration Database is based on confirmation of data existence in appropriate tables in every replica in system. The proper data is confirmed in each replica in tables task and web_object.

Ad 4. Creating task with web object type.

1. To create new task in IBIS user panel (User Panel / Task ordering / Crawler / Single) the form is filled with the following parameters:
   - Start URL = ftp://awxcnx.us/
   - MAX Depth = 2
   - Accept URL = CRAWL_SINGLE_DOMAIN
5.4 Evaluation scenarios and results

Write URL = CRAWL_SINGLE_DOMAIN
Order Ownership = crawling
Result Data Set = CRAWLING_04.TEST_WEB_OBJECT_TYPE

2. Data verification in Configuration Database is based on confirmation of data existence in appropriate tables in every replica in system. The proper data is confirmed in each replica in tables task and web_object_type.

Ad 5. Adding task with related domain.

1. To create new task in IBIS user panel (User Panel / Task ordering / Crawler / Single) the form is filled with the following parameters:
   Start URL = http://not.existing.domain.com/
   MAX Depth = 2
   Accept URL = CRAWL_SINGLE_DOMAIN
   Write URL = CRAWL_SINGLE_DOMAIN
   Order Ownership = crawling
   Result Data Set = CRAWLING_05.TEST_DOMAIN

2. Data verification in Configuration Database is based on confirmation of data existence in appropriate tables in every replica in system. The proper data is confirmed in each replica in tables task and domain.

Ad 6. Creating automatic Crawler task.

1. To create new task in IBIS user panel (User panel / Task ordering / Crawler / Automatic) the form is filled with the following parameters:
   Initial URL Feeder = Google List
   MAX Depth = 2
   Accept URL = CRAWL_SINGLE_DOMAIN
   Write URL = CRAWL_SINGLE_DOMAIN
   Order Ownership = automatic crawling
   Result Data Set = CRAWLING_06_TEST_AUTOMATIC
2. Data verification in Configuration Database is based on confirmation of data existence in appropriate tables in every replica in system. The proper data is confirmed in each replica in tables task, task_instance, domain, web_object, web_object_type and active_job.

5.4.3 Crawling

Crawling [65] is a basic mechanism offered in IBIS system. It is a process of seeking of WWW pages (including FTP) and generation of URL references for founded objects. Crawling in IBIS system is a way of generation of URL references to Internet objects, which are then analyzed by specific Working Agents. There are two types of Crawling in IBIS system:

- Single, in which task is created with only one URL defining an address of page from which Crawling process would start.

- Automatic, which allows to create tasks with references to many pages, which can be defined as a list of URL, or can be downloaded from Internet search engines (Yahoo, Googgle).

After task parameters are defined, the task is created and SeedsAgent is invoked. SeedsAgent generates links according to specified strategies and limitations. These addresses are then parsed by Working Agents called HtmlParserAgent.

Crawling test case is conducted in the system based on the architecture presented in fig. 5.16.

Aim of test

The aim of the following test case is exploration of Crawling process in IBIS system with replicated Configuration Database.

Test schema
5.4 Evaluation scenarios and results

Figure 5.16: Crawling test architecture

The test of Crawling process in configuration with replicated Configuration Database is performed in the following schema:

1. Setting appropriate strategy for IBIS cluster.
2. Creating Crawling task with user interface.
3. Crawling task assignment to SeedsAgent by Dispatcher.
4. SeedsAgent task performing and writing appropriate task end state.
5.4 Evaluation scenarios and results

6. Task results exploring and verification with user interface.


Test details

Ad 1. Setting appropriate strategy for IBIS cluster.

Using Web Interface, IBIS cluster strategy needs to be modified to enable test completion. It is done in administrator panel (Administration Panel / Databases / Strategy).

Ad 2. Creating Crawling task with user interface.

To create new Crawling task the appropriate fields must be filled in IBIS user panel (User Panel / Task ordering / Crawler / Single for Single crawling tasks, User Panel / Task ordering / Crawler / Automatic for Automatic tasks).

Ad 3. Crawling task assignment to SeedsAgent by Dispatcher.

Verification of correctness of task assigning to the system can be performed in administrator panel (Administration Panel / Databases / Active Tasks / Ready Tasks).

Using a list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress) confirm whether the task is appropriately attached to agent SeedsAgent.

Ad 4. SeedsAgent task performing and writing appropriate task end state.

Verify whether the appropriate task is removed and is not moved to Incorrectly Finished, which can be done using the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress).
5.4 Evaluation scenarios and results


Again, using the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress) check whether orders for parsing of pages appear.

Ad 6. Task results exploring and verification with user interface.

The result of work of agent SeedsAgent, which realizes Crawling algorithm, is a set of URL addresses. This result set of URLs is called Data Set. Data Sets can be reviewed, managed or removed from user panel (User Panel / Result Review / Data Sets).

HtmlParserAgent as a result of its work generates a set of attributes for every URL address previously found by Crawler agent. HtmlParserAgent results can be reached as web objects in user panel (User Panel / Result Review / Web Objects).


Data Sets can be removed using Data Sets in user panel (User Panel / Result Review / Data Sets). After remove is completed appropriate message is presented.

Test example

Ad 1. Setting appropriate strategy for IBIS cluster.

Strategy consisting of SeedsAgent and HtmlParserAgent agents is prepared. Configuration with 2 nodes is prepared. In every node runs 1 SeedsAgents agent and 1 HtmlParserAgent agent.

Ad 2. Creating Crawling task with user interface.

To create new task the form in IBIS user panel (User Panel / Task ordering / Crawler / Single) is filled with the following parameters:
5.4 Evaluation scenarios and results

Start URL = http://student.agh.edu.pl/
MAX Depth = 2
Accept URL = .+
Write URL = .+
Order Ownership = crawling
Result Data Set = CRAWLING_07_TEST_FUNCT

Ad 3. Crawling task assignment to SeedsAgent by Dispatcher.

Using administrator panel (Administration Panel / Databases / Active Tasks / Ready Tasks) verification of correctness of task assigning to the system is performed.

With usage of the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress) it is confirmed that task is appropriately attached to agent SeedsAgent.

Ad 4. SeedsAgent task performing and writing appropriate task end state.

Verification whether the task is removed and is not moved to Incorrectly Finished, is done using the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress).


The list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress) is used to check that orders for parsing of pages appeared.

Ad 6. Task results exploring and verification with user interface.

In user panel (User Panel / Result Review / Data Sets) Data Sets are verified. To do it the following data is presented and checked:
5.4 Evaluation scenarios and results

- URLs that are in Data Set,
- Data Set details.

Contents of web objects for sample pages in Data Set are presented with the usage of user panel (User Panel / Result Review / Web Objects), and then it is compared with original.

*Ad 7. Removal of data created by Working Agents.*

Data Sets created on this test case is removed using Data Sets in user panel (User Panel / Result Review / Data Sets). After remove completion it is verified whether data is really removed from database using the same view in user panel.

### 5.4.4 Data Searching

Data Searching is performed to find documents (WWW pages) that contains specified key words. Data Searching test case is conducted in the system based on the architecture presented in fig. 5.17.

**Aim of test**

The aim of this test case is verification of functionality that enables searching of patterns in data stored in the system. Data Searching mechanism in replicated IBIS system is explored in the following test case.

**Test schema**

The test of Data Searching process in configuration with replicated Configuration Database is performed in the following schema:

1. Perform single Crawling.
2. Setting appropriate strategy for IBIS cluster.
5.4 Evaluation scenarios and results

Figure 5.17: Data Searching test architecture

3. Creating task of searching Data Set with user interface.
4. Creating task of searching all Data Sets with user interface.
5. Data Searching task assignment to SearchWVAgent by Dispatcher.
6. SearchWVAgent task performing and writing appropriate task end state.
7. Task results exploring and verification with user interface.
5.4 Evaluation scenarios and results

Test details

Ad 1. Perform single Crawling.

Single Crawling need to be performed. Creating Crawling in precisely described in Crawling test case.

Ad 2. Setting appropriate strategy for IBIS cluster.

Using Web Interface, IBIS cluster strategy needs to be modified to enable Data Searching. It is done in administrator panel (Administration Panel / Databases / Strategy).

Ad 3. Creating task of searching Data Set with user interface.

With the usage of user panel (User Panel / Task ordering / Searching) create a task of parsing a chosen group.

Ad 4. Creating task of searching all Data Sets with user interface.

With the usage of user panel (User Panel / Task ordering / Searching) create a task of parsing all Data Sets.

Ad 5. Data Searching task assignment to SearchWVAgent by Dispatcher.

Verification of correctness of task assigning to the system can be performed in administrator panel (Administration Panel / Databases / Active Tasks / Ready Tasks).

Using a list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress) confirm whether the task is appropriately attached to agent SearchWVAgent.
5.4 Evaluation scenarios and results

Ad 6. SearchWVAgent task performing and writing appropriate task end state.

Verify whether the appropriate task is removed and is not moved to Incorrectly Finished, which can be done using the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress).

Ad 7. Task results exploring and verification with user interface.

SearchWVAgent as a result of its work generates a set URL addresses called Data Set. Data Sets can be reviewed and managed in user panel (User Panel / Result Review / Data Sets). Using this panel it must be verified whether agent SearchWVAgent finished its job correctly.

Test example

Ad 1. Perform single Crawling.

Tasks related to Crawling test case are performed.

Ad 2. Setting appropriate strategy for IBIS cluster.

New strategy assuming invocation of SearchWVAgent agents is confirmed. Strategy consisting of SeedsAgent and HtmlParserAgent agents is prepared. Configuration with 2 nodes is prepared. In every node runs 1 SeedsAgents agent, 1 HtmlParserAgent agent and 1 SearchWVAgent agent.

Ad 3. Creating task of searching Data Set with user interface.

To create new task of single Data Searching the form in IBIS user panel (User Panel / Task ordering / Searching) is filled with the following parameters:

Input Case = crawling
Input DataSet = CRAWLING_07_TEST_FUNCT
Search Query = wordvec: + 0 ibis
Order Ownership = searching
Result Data Set = SEARCHING_01_TEST_FUNCT

Ad 4. Creating task of searching all Data Sets with user interface.

To create new task of Data Searching for all Data Sets the form in IBIS user panel (User Panel / Task ordering / Searching) is filled with the following parameters:

Input Case = crawling
Input DataSet = ALL_DATASETS
Search Query = wordvec: + 0 ibis
Order Ownership = searching
Result Data Set = SEARCHING_02_TEST_FUNCT

Ad 5. Data Searching task assignment to SearchWVAgent by Dispatcher.

Using administrator panel (Administration Panel / Databases / Active Tasks / Ready Tasks) verification of correctness of task assigning to the system is performed.

With usage of the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress) it is confirmed whether Dispatcher appropriately attached the task to agent SearchWVAgent.

Ad 6. SearchWVAgent task performing and writing appropriate task end state.

Verification whether the Data Searching task is removed and is not moved to Incorrectly Finished, which is done using the list of processed tasks in administrator panel (Administration Panel / Databases / Active Tasks / Tasks in Progress).

Ad 7. Task results exploring and verification with user interface.
In user panel (User Panel / Result Review / Data Sets) it is verified that existed Data Sets that are the result of searching performed by agent SearchWVAgent. The correctness of generated data is also confirmed.

### 5.4.5 Strategy management

Replication of IBIS strategies containing common steering parameters which controls system behavior in a dynamically changed environment is evaluated as a part of this research. Architecture used in the following test is presented in fig. 5.18.

![Strategy management test architecture](image)

Figure 5.18: Strategy management test architecture
Aim of test

The aim of this test case is to explore system efficiency of strategy management process in IBIS system with Theta replication implemented.

Test schema

The test of Data Searching process in configuration with replicated Configuration Database is performed in the following schema:

1. Setting initial strategy for IBIS clusters.
2. Running Cluster Supervisor Agents.
3. Measuring response times for strategy access.
4. Verification of clusters strategies with user interface.
5. Performance evaluation.

Test details

Ad 1. Setting initial strategy for IBIS clusters.

Using Web Interface, strategies in IBIS clusters need to be modified. It is done in administrator panel (Administration Panel / Databases / Strategy).

Ad 2. Running Cluster Supervisor Agents.

Run appropriate amount of Cluster Supervisor Agents in remote nodes of the system. To run single process of Cluster Supervisor Agent start_cluster_manager.bat located in IBIS home directory is used.

Ad 3. Measuring response times for strategy access.
5.4 Evaluation scenarios and results

Measure and gather response time for database calls related to the strategy management.

Ad 4. Verification of clusters strategies.

After Cluster Supervisor Agents finished theirs work, verify the correctness of strategies in replicas for particular clusters. To do this compare the strategy contents in strategy and cluster_strategy_property tables.

Ad 5. Performance evaluation.

Performance evaluation is realized as a comparison of the strategy management processes in case of centralized Strategy Database and in case when strategy related data is replicated using Theta replication method.

Test example

Ad 1. Setting initial strategy for IBIS clusters.

Initial strategy assuming invocation of SearchWVAgent agents is confirmed. Strategy consisting of SeedsAgent and HtmlParserAgent agents is prepared. In every node in the system runs 1 SeedsAgents agent, 4 HtmlParserAgent agents and 2 SearchWVAgent agents.

Ad 2. Running Cluster Supervisor Agents.

10 Cluster Supervisor Agents is run per single node of IBIS system. Every Cluster Supervisor Agent performs 10 transactions operating on strategies per second. Within the test 10 up to 200 Cluster Supervisor Agents is run (IBIS environment used in test limited to 20 nodes), which allows to gain the load of
5.4 Evaluation scenarios and results

2000 tps.

*Ad 3. Measuring response times for strategy access.*

On the purpose of the test a special version of Cluster Supervisor Agent software is prepared, which measures the time differences between the start and end of the transaction, and writes the appropriate information to the IBIS system log.

*Ad 4. Verification of clusters strategies with user interface.*

The correctness of strategy contents in `strategy` and `cluster_strategy_property` tables is confirmed.

*Ad 5. Performance evaluation.*

It can be noticed that with the increasing number of Cluster Supervisor Agents, Strategy Database becomes overloaded while maintaining strategy related requests. Performed measures shows that the strategy maintenance for 870 tps (87 Cluster Supervisor Agents generating 10 transactions per sec.) totally saturated used database server not allowing to start new processes in the system, even though no other operation in Configuration Database nor Working Database are performed [52]. Since Cluster Supervisor Agent generate 1 transaction per second, 870 tps for centralized database means that centralized database can handle sessions for up to 870 Cluster Supervisor Agents, without having resources for other activities in the system. The results gained for centralized Strategy Database is shown in fig 5.19 and related table 5.1.

Theta replication approach implementation in configuration with 10 replicas allows to satisfactorily handle increased load related to Strategy Management in IBIS system. The performance results for data replication implementation are presented in fig. 5.19 (10 replicas). Gained tests results show that while the amount of Cluster Supervisor Agents is growing, the system with data replication still operates within an acceptable efficiency limits.
5.4 Evaluation scenarios and results

For loads lower than 400 tps replicated database behaves worse than centralized database, which is caused by additional communication overhead in middleware components. However, for loads starting from 400 tps, centralized database performance starts to decrease significantly, while replicated database response times grows slightly even for loads higher than 1000 tps. CPU average load is around 30% for 2000 tps in replicas of IBIS, while in centralized database it is 100% busy when the load exceeds 800 tps.

The replicated system can handle more concurrent sessions from agents, allowing on invocation of thousands of simultaneously running Cluster Supervisor Agents. Load increased from 100 to 870 tps causes response times of IBIS centralized database grows more than 400% and database become saturated. When IBIS strategies are replicated to 10 database copies in remote locations, the mean response time of replicated database grows only 23% although load changes from 100 to 2000 tps.

Figure 5.19: Strategy management efficiency comparison
5.4 Evaluation scenarios and results

<table>
<thead>
<tr>
<th>Load [tps]</th>
<th>Centralized DB</th>
<th>10 Replicas</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>16.54</td>
<td>24.61</td>
</tr>
<tr>
<td>200</td>
<td>18.63</td>
<td>23.00</td>
</tr>
<tr>
<td>300</td>
<td>22.44</td>
<td>23.82</td>
</tr>
<tr>
<td>400</td>
<td>22.71</td>
<td>23.43</td>
</tr>
<tr>
<td>500</td>
<td>28.34</td>
<td>23.36</td>
</tr>
<tr>
<td>600</td>
<td>30.91</td>
<td>24.09</td>
</tr>
<tr>
<td>700</td>
<td>42.53</td>
<td>23.57</td>
</tr>
<tr>
<td>800</td>
<td>49.64</td>
<td>23.28</td>
</tr>
<tr>
<td>870</td>
<td>66.31</td>
<td>-</td>
</tr>
<tr>
<td>900</td>
<td>-</td>
<td>24.97</td>
</tr>
<tr>
<td>1000</td>
<td>-</td>
<td>25.46</td>
</tr>
<tr>
<td>1200</td>
<td>-</td>
<td>27.25</td>
</tr>
<tr>
<td>1300</td>
<td>-</td>
<td>29.92</td>
</tr>
<tr>
<td>1500</td>
<td>-</td>
<td>27.33</td>
</tr>
<tr>
<td>2000</td>
<td>-</td>
<td>29.86</td>
</tr>
</tbody>
</table>

Table 5.1: Mean response times of transactions for Cluster Supervisor Agent

5.4.6 Method adaptation

IBIS system is an extensive and complex project and changes in its source code is quiet challenging. To make an adaptation of Theta replication method easier for IBIS system a special interface is implemented. This interface is realized by extending IBIS database management class for handling data replication management.

```java
public WebObject insertWebObject(String url) throws DisDatabaseException {
    if (StringUtil.isEmpty(url)) {
        throw new DisDatabaseException("Cannot insert new web object with empty url");
    }
    try {
        WebObject webObject = new WebObject(url);
        webObjectService.insertICNeExists(webObject); // insert in WebObjectServiceICNeExists.java
        return webObject;
    } catch (MalformedURLException e) {
        throw new DisDatabaseException(e);
    }
}
```

Figure 5.20: Listing for inserting web object – centralized database
5.5 Summary

The replication interface created on the IBIS purposes allows on extremely straightforward adaptation of Theta replication method in IBIS. As it is presented in 5.3.4 data management for centralized database is realized through iBATIS framework. Listing presented in fig. 5.20 provides an example of inserting new Web Object to centralized database. Parameters of object WebObject are used during data insertion into database. The same parameters are used for creation of strings containing the set of parameters as it is shown in fig. 5.21. The implemented replication interface allows to submit such prepared strings to the middleware, and in consequence appropriate procedures are executed in replicas.

```java
public String InsertWebObject(WebObject webObject) {
    String strDbCall = "N||insert_web_object(" + webObject.getDomainId() + 
    strDbCall += ", " + webObject.getTypeId() + ", CAST ('" + webObject.getSuffix() + 
    " AS character varying)"");

    // Call stored procedure in replica
    String strDbResult = ExecuteProcedure(strDbCall);

    return strDbResult;
}
```

Figure 5.21: Listing for inserting web object – replicated database

5.5 Summary

Implementation of multi-agent IBIS system using presented Theta replication method satisfies the requirements of efficient Internet scanning process realized in the system. Strategy Management process evaluation confirms the high scalability of the replication method in the real life replication process. Adding new replicas into the IBIS system causes that performance of the replication management process lowers slightly, while on the other hand the overall efficiency of the activities realized in the system grows significantly allowing to provide service with greater amount of agents working at the same time (Cluster Supervisor Agents and related Working Agents). Replicated system can handle more concurrent sessions from agents, allowing on invocation of thousands of simultaneously
running Cluster Supervisor Agents. Performed failure resistance tests in the real life system confirm the replicated system operates properly and is resistant to failures.

Data replication based on Theta replication method allows to preserve previously implemented functionalities, while on the other hand allows to adapt easily Theta method to the requirements of the data replication in IBIS, without the necessity of performing changes in the core of the implemented system.

Higher distances between remote replicas and lowered speed of network interfaces cause the response times of the replicated system are expected to grow. However, in the system with Theta method implemented these response times are expected to grow slowly since the amount of the communicates exchanged between remote locations is reduced considerably (confirmation of the commit from the remote replicas is not sent). Appropriate tests, however, are not yet performed because of the short distances between particular replicas in current IBIS implementation, which cause considerable network delays do not appear in IBIS related infrastructure.
Chapter 6

Conclusions

Theta replication method has been designed for the high availability multi-tier architecture with distributed middleware. The middleware uses its own concurrency control mechanism, called Conflict Prevention algorithm, which enables transactions to be executed with a degree of parallelism. Conflict Prevention algorithm defines transaction processing order in a way that ensures nonconflicting processing. Moreover, it supports execution of transaction with a degree of parallelism, while at the same time guarantees data consistency in replicas (the same results of processed transactions in each replica). Since Conflict Prevention algorithm ensures that conflict can not happen in any replica, which technically is a single database instance, transactions are applied to the system in a way as they would be applied in a system with centralized database. As a result usage of distributed data locks for such data replication is not required, which is confirmed within the tests performed for both implementations of the replication system based on Theta method.

In laboratory experimental evaluation of Theta replication control method shows that Theta method provides scalable and non-conflicting mechanism to keep multiple replicas synchronized. The method ensures high scalability and at the same time provides acceptable efficiency of data replication. Theta approach is suitable for heterogeneous environments which is implied as a possibility of replication implementation in environments with different RDBMS, OS and/or platforms without the necessity of using complex gateways, additional drivers, etc. The replication system is compatible with the most of existing database sys-
tems and since the approach is based on the execution of the stored procedures in remote replicas, it can be used in heterogeneous environments with no difficulty. Data replication based on Theta method allows to preserve previously implemented functionalities of IBIS system, while on the other hand allows to adapt easily the method to the requirements of the data replication in IBIS without the necessity of performing changes in the core of the implemented system.

Implementation of the multi-agent IBIS system based on Theta replication method satisfies the requirements of the Internet monitoring realized in the system. Adding more replicas to the system causes its performance lowers slightly in comparison to the configuration with only two database copies, which allows to provide service with more concurrently working agents and in consequence significantly improves overall system processing performance. Performed fault tolerance evaluation of the IBIS system with replication confirms resistance to failures ofTheta method.

In the author’s opinion the most important achievements of this dissertation are:

- Design of the new data replication method for transactional, multi-node, distributed systems.

- Implementation of the method both in the laboratory environment and on the purpose of information sharing in the operative multi-agent IBIS system.

- Performing wide-ranging positive tests of the proposed method.

Theta replication method has been designed, implemented and evaluated both in in-laboratory environment and in the real life multi-agent IBIS system. Replication software built for the purposes of these implementations can be used for data replication in the systems which need to share data between remote locations. However, additional work is required to be done to make the software usage in the future implementations easier. The following list presents tasks that are planned to be performed in the future:
• Evaluation of the replication method in the environment with less efficient network (higher distance between remote replicas, lower speed of network interfaces).

• Implementation of the universal connection handler that can be used for the majority of the available relational database management systems (until now Theta replication method has been implemented and tested for the limited amount of relational database management systems).

• Packaging the software and preparation of the universal installer to provide easy deployment method of the software as well as the replication configuration.

• Preparation of the web-based repository for binaries and source codes which will act as a centralized location for software packages connected with Theta replication method, and which can be used by software developers to control and manage software releases.
Appendix A

Theta method software deployment

Theta replication system is implemented for both C and Java programming languages, and for different hardware platforms and databases. The wide-ranging evaluation of the implemented method has been performed and built replication software can be used for the purposes of data sharing based on Theta replication method. It is planed to package the software and to prepare universal installer to enable easier deployment of the replication software as well as its configuration.

Attached materials can be used to set up such replication environment as it is used within the performed evaluation of Theta method.

CD content

Description of the elements are placed on the attached CD is presented in fig A.1. Binaries of the IBIS agents equipped with services for communication with middleware replication subsystem as well as libraries providing database services used by IBIS agents are not included on the CD since the author of this dissertation is not the owner of the IBIS system.
Figure A.1: Elements are placed on the attached CD

Content of replication system

Directories

- lib - libraries providing database services used by IBIS agents.
- logs - logs of the middleware components.
- tlogs - middleware transactional logs.

Middleware components for in-laboratory implementation

- cman - connection manager,
- qman - queue manager,
- crm - conflict resolution manager,
- spex - stored procedure executor,
- gidg - global identifier generator.
Libraries for in-laboratory implementation

- \texttt{lib\_qman} - qman queue management,
- \texttt{lib\_ora} - support for Oracle database,
- \texttt{lib\_func} - additional functions.

Middleware components for IBIS implementation

- \texttt{cman.class} - connection manager,
- \texttt{qman.class} - queue manager,
- \texttt{crm.class} - conflict resolution manager,
- \texttt{spex.class} - stored procedure executor,
- \texttt{gidg.class} - global identifier generator,
- \texttt{rcvman.class} - recovery manager.

Communication classes

- \texttt{CmanConnectionHandler.class} - implementation of the listener for cman,
- \texttt{GidgConnectionHandler.class} - implementation of the listener for gidg,
- \texttt{ReceiveMessageInterface.class} - RMI communication interface.

Configuration files

- \texttt{theta.properties} - definition of the main parameters for each of the middleware components (database connection strings, IP, port numbers, etc.),
- \texttt{log4j.cman.properties} - log4j configuration for cman,
• log4j.qman.properties - log4j configuration for qman,
• log4j.crm.properties - log4j configuration for crm,
• log4j.spex.properties - log4j configuration for spex,
• log4j.recover.properties - log4j configuration for rcvman.//

Setting up environment

Example directory structure for IBIS replication implementation is presented in fig. A.2.

Figure A.2: Middleware directory structure for IBIS implementation
Running middleware replication processes

Example of the start of the middleware replication processes in IBIS system is presented in fig. A.3.

```
C:\ibis> java -cp ..;C:\ibis\java\lib\log4j-1.2.14.jar gldgn
C:\ibis> java -cp ..;C:\ibis\java\lib\log4j-1.2.14.jar maan
C:\ibis> java -cp ..;C:\ibis\java\lib\log4j-1.2.14.jar quan
C:\ibis> java -cp ..;C:\ibis\java\lib\log4j-1.2.14.jar cm
C:\ibis> java -cp ..;C:\ibis\java\lib\log4j-1.2.14.jar spaw
```

Figure A.3: Running middleware processes for replication in IBIS
Appendix B

Implementation details

B.1 In-laboratory

The platform compatibility of the implementation of Theta replication method is fulfilled by the development of its code in $C$ language, which compiles and executes as expected on many different hardware and software environments. Every Theta replication middleware component is implemented in $C$ language.

The data replication core software is compiled with the usage of:

- Sun C 5.8 compiler for servers provided by Oracle Sun,
- Gcc 3.4.5 for linux based servers.

Communication between the middleware components is based on the usage of Inter-Process Communication (IPC) mechanisms [91, 102]. IPC is a set of techniques for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC techniques are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.
CMAN middleware process acts as a server listening on a port and waiting for incoming requests from clients. Each of the client’s connections is managed by a dedicated thread to allow clients to establish multiple, concurrent connections. Implementation of the connection handling is based on TCP sockets [102] usage. A TCP socket is defined as an endpoint for communication, and consists of the pair (IP Address, Port). A TCP connection consists of a pair of sockets, which are distinguished by client and server sockets.

The communication between connection manager and global identifier generator is realized with the usage of TCP sockets. There is only one GIDG process for each of replicas in the laboratory implementation of the data replication. This process runs on a dedicated, separate machine, however, it is possible to run it on the same server machine where the other middleware processes are run.

Theta replication method is implemented and tested with both commercial and non-commercial open source relational database systems. The implementation and tests are realized for Oracle 10.2.0.4 and PostgreSQL 8.1.4 relational database systems.

Oracle database connections is maintained with the usage of Oracle Call Interface (OCI) [69] implemented within Oracle Instant Client 10.2. According to Oracle documentation, OCI is the most comprehensive, high performance, native C language based interface to the Oracle Database that exposes the full power of the Oracle Database. OCI provides the foundation on which various language specific interfaces such as Oracle JDBC-OCI, ODP.Net, Oracle Precompilers, Oracle ODBC and Oracle C++ Call Interface (OCCI) drivers are built. OCI is also used by leading Open Source interfaces such as the PHP OCI8 extension, ruby-oci8, Perl DBD::oracle and Phython cx_oracle. OCI being the native interface to the Oracle Database also powers queries, statements and calls issued by the Oracle Database kernel internally. Various Oracle tools such as SQL*Plus, Real Application Testing (RAT), SQL*Loader and Data-Pump are also OCI based. In addition, Oracle In-Memory Database Cache (IMDB Cache a.k.a. TimesTen) supports OCI and Oracle Pro*C/C++ Pre-compilers.

PostgreSQL database communication is based on the usage of C Library libpq [86]. Libpq is the C application programmer’s interface to PostgreSQL. Libpq is a set of library functions that allow client programs to pass queries to the
PostgreSQL back-end server and to receive the results of these queries. Libpq is also the underlying engine for several other PostgreSQL application interfaces, including those written for C++, Perl, Python, Tcl and ECPG.

B.2 IBIS case

To ensure compatibility with the IBIS system based on Java platform, data replication implementation is realized in Java programming language. The Internet monitoring system with data replication implementation is built with the usage of Java compiler in the following version:

Java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode, sharing)

Communication between the middleware components for IBIS system data replication implementation is based on the usage of Remote Method Invocation (RMI) [39, 71]. Java RMI enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. RMI uses object serialization to marshal and unmarshal parameters and does not truncate types, supporting true object-oriented polymorphism. Java RMI is included with Java SE and is available as a separate download for Java ME.

The communication between connection manager and global identifier generator is based on TCP sockets [102] usage. In the implementation for IBIS system there is only one GIDG process dedicated for servers in every site which runs within its own Java Virtual Machine (JAVA VM) and is located in the same server machine where the other middleware processes run. However, it is possible to exclude GIDG from the common server, and invoke within Java VM that runs on the dedicated server.
PostgreSQL 8.1.4 relational database system is used to store data in the IBIS system. The communication with the centralized database used for storing temporary agents data is realized with \textit{iBATIS} framework \cite{ibatis}. IBATIS is a data mapper framework that made it easier to use a relational database with object-oriented applications. Data flow in iBATIS based environment is presented in fig \ref{fig:ibatis}. There are both Java and .Net implementations of iBATIS. To implement database interface relying on the iBATIS data mapper application objects, XML, and SQL are used.

\textit{Theta replication} method is implemented for replication-based information sharing in multi-agent IBIS system. Data in the system is also stored in PostgreSQL 8.1.4 relational database system. The middleware components communicate with data replicas by stored procedures calls, which are realized through the JDBC driver for PostgreSQL database packaged into postgresql-8.4-701.jdbc3.jar \cite{jdbc}. PostgreSQL JDBC driver allows Java programs to connect PostgreSQL database using standard, database independent Java code. It is a Java Type IV implementation. The PostgreSQL JDBC driver provides a complete implementation of the JDBC 3 specification in addition to some PostgreSQL specific extensions.

For compatibility with the IBIS system logging within data replication is realized on the basis of the Log4j technology. Log4j for Java (log4j-1.2.14.jar) is a product included in the Apache Logging Services Project frameworks. The Apache Logging Services Project creates and maintains open-source software related to the logging of application behavior and released at no charge to the public \cite{log4j}. 

\begin{figure}[h]
\centering
\caption{Data flow in iBATIS based environment}
\end{figure}
Appendix C

Test environment details

C.1 In-laboratory

Within the performed experiments data replicas located in remote sites are connected by a Local Area Network. Each server holds a copy of the replicated database and an instance of the middleware.

Figure C.1: Laboratory environment for Theta replication implementation
Clients applications are run on SunFire V490, 4 x UltraSPARC-IV+ 1500MHz, 16GB RAM, Solaris 10. The clients are modeled as the separate processes located in both sites and connect to the nearest replicas. Approximately the overall load is balanced equally between sites, since the number of clients processes and runs is the same in both sites during all of the experiments.

The Network specification: All servers are connected via an Ethernet-LAN running at 1000Mbps.

The relational database systems used in the laboratory experiments: Oracle ver. 9 - 11 and PostgreSQL 8.1.4.

The replication core software is compiled with the Sun C 5.8 compiler for servers provided by Sun, and gcc 3.4.5 is used for Linux servers.

Oracle database communication is maintained with the usage of Oracle Instant Client 10.2.

Since test servers and network are normally used for work during workdays, most of the experiments are conducted only by night hours and at weekends to avoid the influence of the users applications.

Laboratory environment configuration for Theta replication method implementation with 2 replication nodes is presented in fig. C.1. A particular amount of clients, which in fact are dedicated processes simulating clients, is run on two clients servers. Clients submit theirs transactions to middleware and after transactions are processed in database, an answer is sent back to client. The transaction time, which is defined as difference between transaction start and end, is measured within the research tests. Transaction start means a moment just before client sends transaction to middleware, transaction end is a moment after client receives answer from the middleware.

The specifications of the servers used in particular scenarios with 1, 2, 5 and 10 replicas is presented in table C.1.

C.2 IBIS case

Within the tests there are performed numbers of experiments exploring system properties as well as analysis of chosen, particular system use cases. Experiments
A series of Cluster Supervisor Agents are run in every machine to realize particular tasks related to strategy management. Each machine running Cluster Supervisor Agents is treated as an autonomous site and is connected to a separate configuration database (replica). For the test purpose it is run up to 10 parallel processes of Cluster Supervisor Agent to simulate higher load in the test environment with limited amount of machines.

To operate on data related to task management Dispatcher Agents also work on data stored in replicas. Each Dispatcher Agent connects and operates on dedicated Configuration Database replica.

In the most of research tests Dispatcher Agents and Cluster Supervisor Agents are connected to a dedicated, local replicas. This allow to verify system behavior with larger number of replicas. However, Dispatcher Agents and Cluster Super-

<table>
<thead>
<tr>
<th>Number of replicas</th>
<th>Hardware configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td>2</td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td>5</td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td>10</td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>2 x Intel Xeon 2.6GHz, 4GB RAM, RedHat EL 5</td>
</tr>
<tr>
<td></td>
<td>NServer ME420 G4, 2 x Intel Xeon 2.00GHz, 4GB RAM, RedHat EL 4</td>
</tr>
<tr>
<td></td>
<td>NServer ME420 G4, 2 x Intel Xeon 2.00GHz, 4GB RAM, RedHat EL 4</td>
</tr>
<tr>
<td></td>
<td>NServer ME420 G4, 2 x Intel Xeon 2.00GHz, 4GB RAM, RedHat EL 4</td>
</tr>
<tr>
<td></td>
<td>Sun Fire 480R, 2 x UltraSparc III+ 1050MHz, 4GB RAM, Solaris 10</td>
</tr>
<tr>
<td></td>
<td>Sun Fire 480R, 2 x UltraSparc III+ 1050MHz, 4GB RAM, Solaris 10</td>
</tr>
</tbody>
</table>

Table C.1: Hardware configuration – in-laboratory tests

are conducted on several machines working concurrently.
A series of Working Agents are run in every machine to realize particular tasks. In the test conducted on the purpose of this research it is assumed that Working Agents share common, centralized Working database.

Transaction times for IBIS system are measured exactly in the same way as within the laboratory tests.

Sample configuration presenting test environment for IBIS system with data replication is presented in fig. C.2. Each replica has the number of IBIS agents attached and the middleware manage transactions and executes them in data replicas on behalf of those agents.

Details of hardware configuration for IBIS test environment are presented in table C.2.
## C.2 IBIS case

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of servers</td>
<td>1 - 20</td>
</tr>
<tr>
<td>Operating system</td>
<td>Windows XP SP3</td>
</tr>
<tr>
<td>Virtual machine configuration</td>
<td>VMware virtual machine with 2CPU and 2GB RAM running on Intel Core2 Quad CPU 4GB RAM, Windows 7 1 Virtual Machine per server</td>
</tr>
<tr>
<td>Database</td>
<td>PostgreSQL 8.1.4</td>
</tr>
</tbody>
</table>

Table C.2: Hardware configuration for IBIS replication tests
References


REFERENCES


REFERENCES


