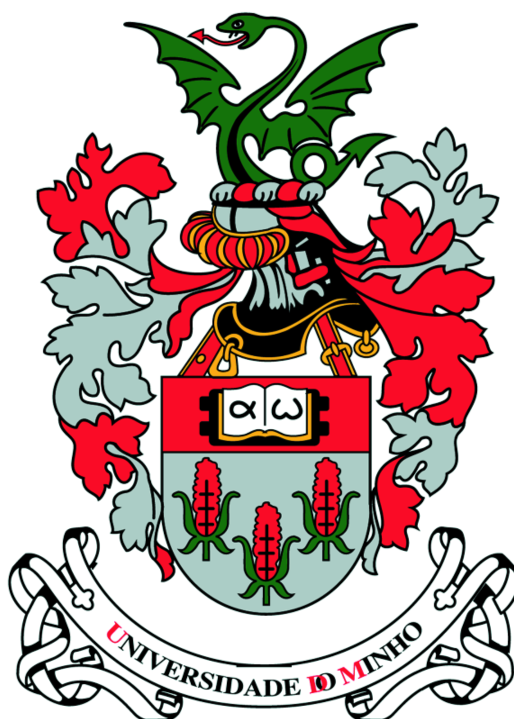
Evaluation of Group-based Database Replication Using Centralized Simulation

Luís Manuel Oliveira Soares



Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Informática, elaborada sob a orientação de José Orlando Pereira

Departamento de Informática
Escola de Engenharia
Universidade do Minho
Braga, 2006

Partially funded by GORDA (FP6-IST2-004758) project.

Resumo

Replicação de bases de dados baseada em comunicação em grupo apresenta-se como uma nova solução para promover sistemas de bases de dados que aliam fiabilidade a elevada disponibilidade. Apesar de existirem algumas soluções de replicação sugeridas neste sentido, não é claro em que situações estas são indiscutivelmente aplicadas de de maneira a optimizar o desempenho. Esta ambiguidade decorre da ausência de um método comum e representativo de avaliação que produza resultados comparáveis para todas as soluções.

Neste contexto, o trabalho apresentado nesta tese resulta numa avaliação exaustiva, em diversos ambientes simulados, de múltiplas implementações de protocolos, de uma forma em que os resultados obtidos são efectivamente comparáveis. A solução proposta oferece a possibilidade de injectar transparentemente implementações reais em ambientes de simulação. Desta forma, é criada uma plataforma de testes extremamente flexível, a qual consegue reproduzir de forma muito próxima os ambientes equivalentes do mundo real. A utilidade desta plataforma é demonstrada através de um estudo que incide em implementações de diversos protocolos de replicação recorrendo a comunicação em grupo. A avaliação é conduzida em diferentes infra-estruturas de rede assim como na presença de faltas.

Abstract

Group-based replication is arising as a new trend to provide dependable, available and reliable database management systems. Although several solutions have been proposed, it is not clear which is the best for each application scenario, as evaluation methods used are not representative or do not produce comparable results.

In this context, the work presented here results in a thorough and comparable evaluation of multiple protocol implementations in diverse simulated environments. The approach allows the tester to transparently embed real implementations within a realistic simulation, creating this way a flexible testing framework that is shown to accurately mimic real world scenarios. The usefulness of the approach is demonstrated by evaluating and comparing several group-based database replication protocols under different network and application scenarios, as well as in the presence of faults.

Acknowledgements

I would like to specially thank my beloved wife, Raquel, for her permanent support, and my adorable son, Gonçalo, who happened to be born during my work in this thesis. They were the ones that suffered the most with my absence, stress and occasional bad mood. Raquel and Gonçalo, I love you both.

I would also like to thank my parents, who always incited me to keep on with my studies, and my sister for always being there when I needed. I know they have sacrificed on my behalf. This is theirs too.

A special thanks to my supervisor, Orlando, who never turned his back on me, always coming up with good suggestions, leading me invariably into the right direction. His guidance was most crucial.

Another special thanks to Rui Oliveira. He is the one who made this possible by inviting me to work in the distributed systems group, back in 2002.

For the rest of my research team, António Sousa and Alfrânio Correia Jr., a big thank you. They were always willing to discuss any matter related to the work in this thesis. Their comments were always most valuable. Thanks to Nuno Lopes, who had his working spot right next to me. He never rejected help of any kind. Thanks to Luciano Rocha, José Pedro, Vitor Fonte, Carlos Baquero, Paulo Sérgio and Francisco Moura, the rest of the Distributed Systems Group members.

Finally, thanks to everyone that read this thesis and contributed with corrections and critics.

Contents

1 Introduction **1**

1.1 Problem Statement 3

1.2 Contributions 4

1.3 Outline 4

2 Evaluation of Group-based Database Replication **6**

2.1 Replication Using Group Communication 6

2.1.1 System Model and Group Communication 6

2.1.2 State Machine Replication 8

2.1.3 Primary-Backup Replication 9

2.2 Group-based Database Replication 11

2.2.1 Relational Databases and Transactions 11

2.2.2 Conservative Replication Protocols 12

2.2.3 Optimistic Replication Protocols 13

2.3 Performance and Dependability Evaluation 16

2.3.1 Testing Environments and Benchmarks 16

2.3.2 Simulation 17

2.3.3 Evaluation Framework 19

3 Centralized Simulation **23**

3.1 System Architecture 23

3.2 Simulation Model 24

3.2.1 Workload Model 24

3.2.2 Transaction Processing Model 26

3.2.3 System Under Test 28

3.2.4 Network Model 29

3.3 Simulation Kernel 30

3.3.1 Real-Time 30

3.3.2 Client/Server Utility Classes 31

3.4 Model Calibration 33
3.4.1 CPU 34
3.4.2 Storage 35
3.5 Validation 37

4 Case Study **40**
4.1 Motivation 40
4.2 Scenarios 41
4.3 Results: Optimistic vs Conservative 43
4.3.1 Coarse Grain 43
4.3.2 Fine Grain 45
4.3.3 Snapshot Isolation 46
4.3.4 Wide Area 48
4.4 Results: Fault-injection and DBSM 50
4.4.1 Correctness and Performance 51
4.4.2 Performability 53
4.5 Simulation Performance 56

5 Conclusion **57**
5.1 Future Work 59

A Distribution Parameter Estimation **60**

B Real vs Simulation Model **62**

List of Figures

2.1 State Machine Replication. 9
2.2 Primary-backup replication. 10
2.3 Conservative replication protocols (CONS). 12
2.4 Optimistic replication protocols 14
2.5 Real implementation: Client/Server interaction UML diagram. 19
2.6 Simulation: Client/Server interaction UML diagram. 20
2.7 Executing simulated and real jobs. 20
2.8 Scheduling events from real code. 21

3.1 Simple overview of the system architecture. 24
3.2 Simulated transactions and replicas interaction. 27
3.3 Simulated client calls server real implementation. 31
3.4 Client real implementation calls simulated server). 32
3.5 *XLogWrite* calls as a function of the number of items accessed. 36
3.6 Validation of the centralized simulation runtime. 37
3.7 Simulation vs Real comparison. 39

4.1 Local area network configuration. 42
4.2 Wide area network configuration. 43
4.3 Performance measurements in a LAN with coarse granularity. 44
4.4 Detailed profiling in a LAN with 270 clients. 45
4.5 Performance measurements in a LAN with fine granularity. 46
4.6 Performance measurements in a LAN with snapshot isolation. 47
4.7 Performance measurements in a WAN. 48
4.8 Handling concurrent transactions. 49
4.9 Detailed profiling of the Neworder transaction in a WAN. 50
4.10 Certification latency (fault injection). 52
4.11 Probability of committing a Neworder transaction within the period of time τ. 54

4.12 Performance comparison. 55
4.13 Ratio of time required to run the simulation vs. simulated time interval (9 servers/increasing number of clients). 56

A.1 Processing time fitting. 61

B.1 Transaction performance comparison. 63

List of Tables

3.1 Transaction types in TPC-C. 25
3.2 Size of tables in TPC-C. 26
3.3 System specifications. 33
3.4 CPU Times distributions (nanoseconds). 34
3.5 Storage simulation model parameters. 36
3.6 Simulation vs Real: TPM and latency results. 38

4.1 Definition of coarse conflict classes for each transaction type in TPC-C. . . 43
4.2 Types of faults injected. 51
4.3 Protocol CPU usage (%). 52
4.4 Abort rates with 3 sites and 1000 clients (%). 53

Chapter 1

Introduction

Information availability becomes critical nowadays mostly due to the data-intensive applications which populate the Internet. Recent studies claim that the volume of generated information is increasing at a rapid pace, in fact some point out that humankind will generate more original information over the few years to come than the amount that was created in the previous 300,000 years combined [Cor01]. These throughput requirements, drive research towards distributed and replicated architectures, providing solutions that enable high availability through network shared contents. Such systems, require efficient storage capability as well as easy retrieval and outstanding processing capabilities. Combining networked storage and processing entities, enables data distribution and/or replication, reduces wasted storage capacity and backup inconveniences as well as increases data availability.

In this context, distributed middleware is showing an exponential growth in the recent years. This is mostly because the *dawn of the Internet* is coming to an end and it is starting to undergo a maturation process. It started moving away from the initial chaos in direction of a more delineated and middleware oriented structure. This trend influences the design and architecture of new applications which more and more are developed with Internet interaction in mind. The distributed nature of applications requires data to be constantly available, dependable and easily retrievable. This leads to replication, which is a mean to assure high availability and dependability. Applications usually rely on a database to store their information, therefore, problems associated with replication arise not by replicating the application, but by replicating data that applications deal with. Such problems may vary a lot depending on, among others issues, the communication infra-structure, the structure and partitioning of data, the number of replicas, the consistency criteria required and how fault-prone is the environment.

Applications store data in information containers known as databases. The informal definition of a database may sound quite similar to the following: *"A database is a collection of data items with a very strict internal organization, which is determined by inter-items*

relationships". While not being entirely precise, this definition is a good starting point and provides a general picture of what a database really is.

In a database, data handling follows a transactional logic. The *Transaction* concept encompasses many actions on the every-day life. Whenever one makes a phone call or buys something at the grocery shop, a transactional activity is engaged. In both cases some resources are allocated. In the former, the call setup establishes a connection allowing the conversation. In the latter the clerk is kept busy fulfilling the customer requests. Also, a state change is performed either because a virtual circuit is created to route the voice transmission or some goods are now in the customer bag instead of the grocery shelves. Note that these actions are done in a single step. When buying something, one cannot receive the good and go away without paying, because that would be a violation of the law. Until the grocery seller receives the money, the goods are not property of the consumer. It is this atomicity in the buying action that makes trade possible, or in other words, it makes transactional activity possible.

When it comes to transactional databases, transactions have one of two possible outcomes: *commit* or *abort*. Committed transactions complete successfully, cannot be aborted and have their updates persistently written into the database. On the other hand, transactions that abort do not have their updates persistently written into the database, and from the data point of view, the system behaves as none of them had actually ever happened [BHG87, CB02]. Aborts are, either explicitly, upon a request from the client (mostly in interactive transactions), or implicitly, due to database internal transaction management actions (e.g., solving deadlocks).

Transactions are actually a good starting point to head towards replication. Since data becomes altered due to the execution of transactions, keeping all replicas coherent is as simple as letting every replica becoming aware of every transaction execution. This apparent simplicity hides a complex set of processes. For instance, every replica must agree on the changes to its local copy and on which order they are applied. Replicated databases exhibit higher complexity when compared to traditional centralized backends. In order to make it all work, more machines have to function correctly and extra processing is needed to accommodate the operations requested by the replication protocols. The coordination among replicas also introduces a certain degree of complexity to system, which without the proper consideration may easily break the replication process. Furthermore, concurrent transactions started in different replicas may originate conflicts leading to performance degradation. Conflict detection is driven by the consistency criteria adopted and the precision of the conflict detection process. Relaxed correctness criteria often improves performance, at the cost of sacrificing some consistency. On the other hand, conservative criteria does not allow any inconsistency but may incur into unbearable performance penalties as well as it often also requires more interaction among replicas. When it comes to detecting conflicts, fine grained conflict detection produces larger messages and minimizes conflicts when compared to coarse grained. There may

be scenarios that do not conceive fine grained control, either because such might require unavailable bandwidth or the processing time required to execute the consistency checks may not meet a real time constraint value imposed by the system. On the other hand, when coarse grain is adopted, some scenarios may exhibit a high rate of false conflicts rendering the approach impractical.

1.1 Problem Statement

Current evaluation methodologies of replication protocols reveal themselves limited. The existent studies report to analysis conducted in local area networks or in small scale scenarios. The fact is that there are few chances to put together a wide area testing environment because that is not cost-effective, does not provide full control over the environment variables and it does not enable deterministic testing. Furthermore, although in local area environments there is more control over the environment, there is still no deterministic execution and, as in the wide area, a full deployment of the software is also required. These issues must be addressed when researching, developing, evaluating distributed replication protocols. Realistic and controlled performance evaluation of distributed and replicated applications is still very hard to achieve, specially if one considers large scale scenarios and fault injection. Although there are few testbeds [ACM06, Fac06, Uni06] that enable evaluation of distributed systems, they all focus a small portion of the spectrum of possible problems.

Pure simulation models may be a solution to this problem, but creating abstract models from real implementations is not always possible or even desirable. Furthermore, in an evaluation scenario, one is interested in evaluating the real solution and not an abstraction that mimics the original behavior.

The problem extends itself when there is the need for testing early and often during the development cycle of a new replication protocol. The researcher/developer finds himself without the possibility to abstract the irrelevant parts of the system, focusing on the protocol and still conduct a realistic evaluation. For instance, when evaluating a database replication protocol, the database engine and the network could be abstracted as realistic simulation models, since they are not really part of the problem, and the protocol should be the only real implementation. This methodology is known as incremental development. In this context, the development process is backed by an abstract simulation model comprised of multiple sub-models that get replaced by real implementations as these become available. The problem here is having the simulation execution to play nicely with the real execution. Whenever real code execution takes place, the time spent on it must be accounted in the simulation time line. This is the case of the approach of centralized simulation proposed in CESIUM [AC97]. Nonetheless, one cannot reuse CESIUM implementation because it defines very specific models that are not completely

suitable for evaluating database replication protocols. In this context, time accounting cannot be done in an *ad hoc* fashion because it would result in a time consuming and error prone process.

1.2 Contributions

Briefly, this thesis proposes a framework that enables realistic evaluation of database replication protocols by combining abstract simulation models and real implementations in a centralized environment [AC97]. It takes advantage of a centralized execution environment which provides means to embed implementations of protocols into simulation. The framework allows changing the scenarios without having to rewrite any source code, by requiring only modification to configuration files. An evaluation case study, using this framework, is presented in which three group communication replication protocols are profiled. The study brings forward conflict related issues and provides a solution as well as evaluates the protocols in local and wide area networks. It also assess the performance degradation when there is fault-injection.

List of contributions:

SSF Extensions for Centralized Simulation

A centralized simulation framework which is based on the SSF specification. It provides transparent means for injecting real implementations prototypes into simulation and synchronize simulation clock with real execution.

SSF-based Database Simulation Model

A database simulation library written in Java. It provides several simulation models mimicking the execution of a real database. Calibration and validation conducted ensures coherent behavior when compared to a real system;

Protocol Evaluation

The case study presents analysis of the behavior of the optimistic and conservative protocols under different conditions. These ranged from relaxed to strong correctness criteria processes, local area to wide area networks and from faultless to faulty environments.

1.3 Outline

This thesis is structured as follows: Chapter 2, introduces the distributed system model considered, brings forward issues related to database replication evaluation and presents the notion of centralized simulation; Chapter 3, discusses the implementation of a simulation framework and its application to the study of database replication protocols; Chapter 4, demonstrates the application of the simulation framework by presenting a case

study that evaluates three distinct synchronous database replication protocols; Finally, Chapter 5, concludes the thesis.

Publications

Portion of the work presented in this thesis has been previously published in the form of conference and workshop papers:

- A. Sousa and J. Pereira and L. Soares and A. Correia Jr. and L. Rocha and R. Oliveira and F. Moura. Testing the dependability and performance of GCS-based database replication protocols. In *Proceedings of The International Conference on Dependable Systems and Networks*. 2005 (DSN'05).
- A. Correia Jr. and A. Sousa and L. Soares and J. Pereira and R. Oliveira and F. Moura. Group-based replication of on-line transaction processing servers. In *Dependable Computing: Second Latin-American Symposium*. 2005 (LADC'05).
- L. Soares and J. Pereira. Experimental performability evaluation of middleware for large-scale distributed systems. In *7th International Workshop on Performability Modeling of Computer and Communications Systems*. 2005 (PMCCS'05).

Chapter 2

Evaluation of Group-based Database Replication

This chapter introduces database replication protocols based on group communication and discusses techniques to evaluate their performance. A brief discussion of group communication and general purpose replication techniques provides the background.

2.1 Replication Using Group Communication

Reliable and high availability software solutions are achieved mostly using replication. It ensures fault-tolerant, dependable systems. This is often done by making use of group communication primitives, which highly reduce the complexity of the procedure, making the implementation really simple.

2.1.1 System Model and Group Communication

A distributed system is modeled as a set of sequential processes, $P = \{p_0, p_1, ..., p_n\}$, communicating using message passing. This set of processes is generally known as the *group*. Whenever a process chooses to send a message to all other processes it addresses that message to the logical group address. The system is asynchronous, hence no assumptions are made regarding the time a message takes to be transmitted or processed. When the group changes, each process installs a new *view*, the new set of currently correct processes, and each member of the current view becomes aware of the composition of the group. *Crash-stop failures* are assumed hence, if a process crashes, it never rejoins the group.

Processes are affiliated to the group and a membership service [gcs01, BJ87] takes care of process affiliation. Moreover, the membership service is responsible for keeping track

of operational and mutually reachable processes. Groups are considered to be dynamic therefore processes may leave the group by expressing voluntarily their will to do so, or may even leave the group unexpectedly. In both cases, the service is responsible for acknowledging the group change, either by being explicitly warned or by detecting the failed member. A set of *group communication services* is assumed to exist, which provides some guarantees on message exchanging. For instance, the system may need to ensure reliable or even atomic message delivery. In some cases message transmission and group membership are tightly related, as is the case of virtual synchrony. Some systems additionally require messages to be delivered at each site in the same order. In this context, two group communication primitives must be addressed: $TOCAST(m, g)$ and $VSCAST(m, g)$.

Total order multicast ($TOCAST$), guarantees that any message, m, sent to a group, g, $TOCAST(m, g)$, is delivered in the same order at every member of group g. This primitive is defined accordingly to the following properties [DSU04]:

Agreement, if a process, p, $TOCAST(m, g)$, and a correct process p' delivers m then every correct processes in g also delivers m;

Total Order, consider any two processes p and p'. Such processes $TOCAST(m, g)$ and $TOCAST(m', g)$, respectively. If two correct processes q and q' deliver m and m', then q delivers m before m' if and only if q' delivers m before m';

Validity, if a correct process, p, $TOCAST(m, g)$, then a correct process, p', eventually delivers m;

Integrity, every correct process q, delivers any message m at most once and only if it was previously $TOCAST(m, g)$ by a process p belonging to group g.

Virtual synchronous multicast ($VSCAST$), allows processes to synchronize messages upon a group change. It is view-oriented meaning that actions like sending and receiving messages are done within the limits of a view context. $VSCAST(m, g)$ satisfies the following properties [gcs01]:

Delivery Integrity, if a process q delivers a message m, then message m was previously $VSCAST(m, g)$ by a process p;

No Duplication, if a process q delivers m and m', then $m \neq m'$;

Sending View Delivery, if a process, q, delivers m in view V, and process p $VCAST(m, g)$ in view V', then $V = V'$;

Virtual Synchrony, if processes p and q install two consecutive views, V and V', then any message delivered by p in V is also delivered by q in V;

Termination of Delivery, if a process p $VSCAST(m, g)$ in view V, then each member q of V either delivers m or installs a new view V'.

These two group communication primitives, although presented separately, are not mutually exclusive. In fact in the implementations presented in this thesis, $TOCAST$ is built on top of $VSCAST$ and a message sorting service [SPS$^+$05].

2.1.2 State Machine Replication

In the state machine approach to replication all replicas execute the same set of actions in the same order achieving this way the same output [Sch93, WPS$^+$00, GS96]. This technique is also known as *active replication*. Since every replica actively handles requests, without contacting any of the others, there is no need for a centralized control during execution. Figure 2.1, depicts the execution flow of a request. Horizontal lines correspond to each process time-line. Arrows between time-lines correspond to messages exchanged and wiggly lines over time-lines are actions taken by each process. Whenever a client issues a request it addresses it to a group of replicas (p_1, p_2, p_3). Each replica in the group executes the request and then replies back to the client. Clients wait for the first reply to arrive, after which, they assume that the execution has ended.

This replication scheme relies on three assumptions: *i*) replicas receive the same messages in the same order; *ii*) requests are executed deterministically; and *iii*) all replicas start from the same initial state.Given *i*), *ii*) and *iii*) consistency is safeguarded.

This strategy requires each replica to behave exactly like the others and replicas going off-line is fully transparent to clients. As long as there is at least one correct replica, the client does not notice performance degradation [WPS$^+$00]. On the other hand, having every replica to perform exactly the same execution, may be pointed out as waste of resources as well as deterministic execution is very often difficult to achieve. Imagine that scheduler processes, at different sites, decide differently on the order of execution of requests. This is problematic when there are concurrent requests. In this situation global state starts to diverge because some replicas may have decided to perform several updates on the same item in different order. A workaround to solve this problem is to handle requests sequentially and transform non-deterministic into deterministic instructions, which in some cases may be difficult or, even worse, impossible to achieve. In the latter situation, this replication strategy is not eligible at all.

Since every replica must deliver requests in the same order, clients make use of the $TOCAST$ primitive when addressing requests to the group. This way, concurrent requests are serialized and concurrency does not become a problem if all replicas respect the given order, when handling the requests.

Since this is a recognized replication technique, its adaption to database replication

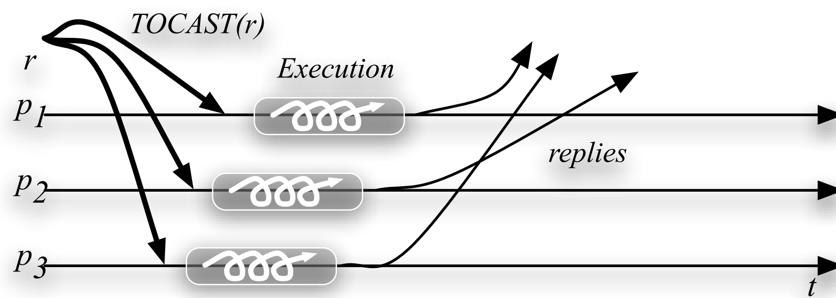


Figure 2.1: State Machine Replication.

has been proposed [PGS98, AT02, WPS+00]. But this technique still suffers from the problems mentioned previously. Non-deterministic instructions must be handled in a way that all database replicas reach the same final state after executing them. This may not be easy to achieve even if one avoids such kind of statements. Nonetheless, achieving determinism at instruction level is still not enough, one needs determinism in the transaction scheduler inside the database engine. For instance, consider two concurrent and conflicting transactions. The deadlock detection algorithm at different replicas may decide differently on which transaction ends up succeeding. There may even be the case in which the transaction scheduler decides to completely ignore the delivery order and commit transactions in its own determined order.

2.1.3 Primary-Backup Replication

Primary-backup, also known as passive replication [BMST93], assumes the existence of one and only one *master* replica, the *primary copy*, and a set of *backups*. In most cases it is irrelevant which replica is the master. Figure 2.2, depicts the protocol execution steps. Arrows between horizontal lines (time-lines) represent messages, wiggly lines represent execution and the newly introduced symbol, double down arrows over time-lines, represent installing the updates. In this figure, the master (p_1), is responsible for interacting with the client, handle its requests and send the updates to the backup replicas (p_2 and p_3). Backups receive only the updates and not the invocation. In fact, these replicas do not perform the execution at all, instead they just install the updates. It becomes clear that p_1 is the only replica executing the request, while p_2 and p_3 keep their state coherent with p_1's state by installing the updates it propagates. Finally, after receiving the acknowledgement of each backup, p_1, replies to the client, terminating the entire operation. Note that this strategy eliminates non-determinism issues, by avoiding multiple execution.

This replication strategy implies two assumptions. *i*) delivery of messages sent from the primary copy to the backups must be atomic and ordered. This is solved, using FIFO (*first-in-first-out*) communication between master and backups. The master assigns

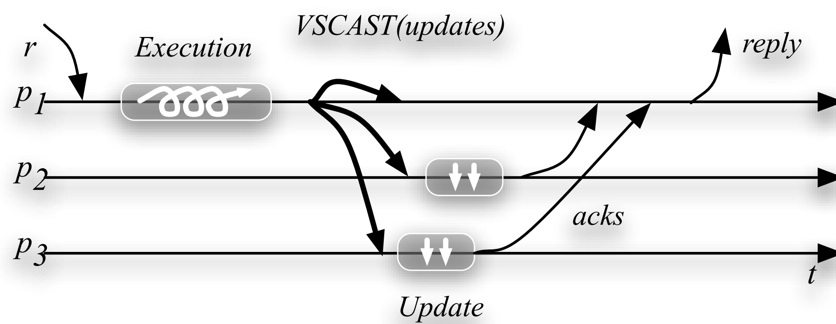


Figure 2.2: Primary-backup replication.

sequence numbers to the update requests, while backups buffer messages arriving out of order and only apply them when their sequence number is reached. *ii*) Whenever the primary copy fails, the backups must agree on which replica takes over the master role and on which messages are yet to be delivered. The election of the new master and message stabilization requires that each backup decide exactly on the same values, and no new master should be elected before all backups agree on the messages delivered.

The agreement property becomes very important in case a master fails. In detail, it may fail: *i*) during the execution of the request; *ii*) while or after sending the update messages, but before replying to the client; or *iii*) after sending the reply to the client. Situations *i*) and *iii*) are not much harmful. In the former, the client needs to reissue the request and in the later the failure is completely transparent to the client. The worst case scenario is situation *ii*), because it leads to inconsistency among replicas. Note that at this point some of the replicas may have received the update messages and others might not. If this is the case replicas begin to diverge. Hence, agreement on message delivery becomes essential because, it ensures that for each message sent from the faulty master, either all or none of the correct replicas deliver it. Moreover, messages sent by the failed master are to be delivered before any message from the new master is delivered. Finally, the last issue taken into consideration relates to the election of a new master. If a master fails, a new leader must be chosen from left correct backups. The election requires that all replicas agree on the new replica chosen.

In this context, VSCAST is a suitable communication primitive for passive replication [GS97] because, it assures agreement on message delivery as well as message stabilization whenever there is a view change. No message from a previous view is delivered in the new view. Furthermore, when a new view is installed every process becomes aware of the group composition, and a new master may be elected. Ultimately, processes may decide which is the new master, by choosing the one with the lower id.

As it happened with the active replication strategy, this one has been also proposed to perform database replication [WPS+00]. Although it does suite database replication it

is sometimes pointed out as being a suboptimal strategy. Having only one master replica serving clients transactions and a set of backups just waiting for updates, is often considered as a waste of resources. This may be critical when deciding whether to replicate a database or not, since such decision is still very performance driven.

2.2 Group-based Database Replication

The performance of database replication using group communication can take advantage of transactional and relational semantics to improve on general purpose active and passive replication approaches. In contrast with traditional database replication approaches, group-based replication protocols take advantage of the specific properties of group communication primitives, such as total ordering and agreement on message delivery, to eliminate the possibility of deadlocks, reduce message overhead and increase performance. Such protocols have been focusing systems that manage data updated frequently and require high availability and short response times. In this context, two kinds of protocols, specifically addressing *On-line Transaction Processing* (OLTP) systems, have recently been the subject of much attention of both theoreticians and practitioners [Ped99, PGS98, PMJPKA00, SPMO02]. They differ mainly in whether transactions are executed conservatively or optimistically. In the former, by *a priori* coordination among the replicas, it is assured that when a transaction executes there is no concurrent conflicting transaction being executed remotely and therefore its success depends entirely on the local DBMS. In the latter, execution is optimistic, each replica independently executes its locally submitted transactions and only then, just before committing, sites coordinate and check for conflicts between concurrent transactions.

2.2.1 Relational Databases and Transactions

A relational database $DB = \{R_1, \ldots, R_s\}$ is a set of relations, $R_i \subseteq D_1 \times \ldots \times D_q$, defined over data sets not necessarily distinct. Each element $(d_1, d_2, ..., d_q)$ of a relation R_i is called a tuple and each d_i is called an attribute. To uniquely identify each tuple of a relation, the existence of a minimum nonempty set of attributes, called the *primary key* is assumed to exist. On certain occasions there may be several distinct sets of nonempty attributes suitable for becoming the primary key. These are named *candidate keys* and only one of them becomes the primary key [OV99].

Tuples are added, removed and changed on user requests. In general, the user initiates a transaction and, within its boundaries, update, delete or insert statements are issued to the DBMS. This logical unit of work, satisfies four properties, often known as ACID [BHG87], namely, *atomicity*, *consistency*, *isolation* and *durability*. The *atomicity property* states that a transaction either ends successfully and its changes are applied into the

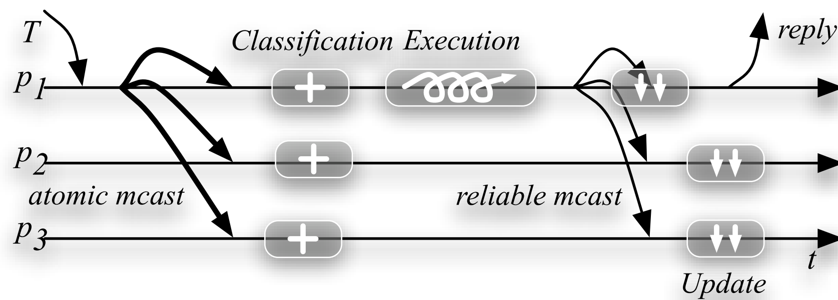


Figure 2.3: Conservative replication protocols (CONS).

database, or, on the contrary, it fails and the database is not altered in any way. The *consistency property* specifies that transactions are responsible for database transitions between consistent states. Apart from programming errors, no database should reach an inconsistent state because of a transaction execution. The *isolation property* determines complete independence between transactions while they are in the execution stage. Finally, *durability property* ensures that the result of a committed transaction becomes persistent and is not lost in case of a failure.

During execution, transactions query and sometimes update the database, hence they read and write tuples. The set of primary keys obtained from the tuples read is named the transaction *read set* (RS). Similarly, the set of primary keys obtained from the tuples written is named the transaction *write set* (WS). Together they from the *basic set* (BS) [OV99] of a transaction, ($BS = WS \cup RS$). Furthermore, the set of tuples read and the set of tuples written are designated by *read values* (RV) and *write values* (WV), respectively. Transaction execution may consist of a plain set of instructions or even comprise another transactions. In the former case, transactions are named *flat transactions* and have a single start and a single end point. In the latter, are named *nested transactions* and inside their boundaries, another transactions exist, named sub-transactions. Despite these two execution models, in this thesis, transactions are always reduced to RS, RV, WS, WV, no matter its execution approach.

2.2.2 Conservative Replication Protocols

In the conservative approach to group-based database replication, data is *a priori* partitioned in conflict classes. Each transaction has an associated set of conflict classes (the data partitions it accesses) which are assumed to be known in advance. While the conflict classes for a transaction can be determined at runtime, this requires knowing the whole transaction before its execution, precluding the processing of interactive transactions.

When a transaction, t, is submitted (Figure 2.3), the replica handling the request issues a $TOCAST(m, g)$. Message m contains, t's identification and conflict classes. Each

replica has a queue associated with each conflict class and, once delivered, a transaction is classified and enqueued in the queues matching the requested classes. As soon as a transaction reaches the head of all of its conflict class queues it is executed and removed from the queues. Transactions are executed by the replica to which they are submitted. When isolated conflict classes exist, dedicating a distinguished replica to the execution of all transactions of such classes, results in a faster processing of those transactions [PMJPKA00].

When the commit request is received, the outcome of the transaction is reliably multicast to all replicas along with the replica's state changes and a reply is sent to the client. Each replica applies the remote transaction's updates with the parallelism allowed by the initially established total order of the transaction.

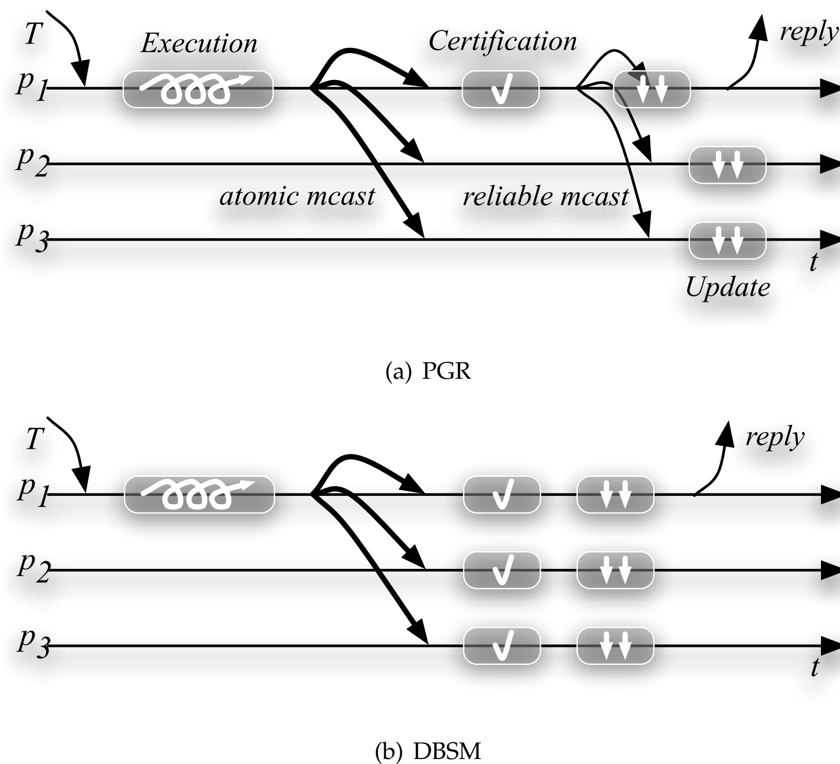
Conflicting transactions are executed sequentially. Clearly, conflict classes have a direct impact on the performance. The lesser the number of transactions with overlapping conflict classes, the better the interleave among transactions. Conflict classes are usually defined at the table level but can have a finer grain at the expense of a non-trivial validation process to ensure that a transaction does not access conflict classes that were not previously specified.

It is worth noting that, despite the use of a multi-version database engine, since conflicting transactions are totally ordered and executed sequentially, the protocol ensures 1-copy-serializability as long as transactions are correctly classified by the application. Relaxing the correctness criterion to snapshot-isolation would simply require the reclassification of the transactions by the application.

Conservative protocols have been introduced and initially evaluated in a cluster environment [PMJPKA00]. It is thus not obvious how they react to different network latencies (e.g., LAN and WAN). Increasing network latency should harm the transaction execution and degrade performance. Another interesting issue to evaluate is the granularity of conflict classes. Should conflict classes be determined a table level or even more fine grained, like tuple level. When defining conflict classes one must consider the trade-off between performance and complexity. Choosing tuple level conflict classes introduces complexity at the application level as well as it is very complicated to predict which tuples the transaction is going to manipulate. On the other hand, if one is able to use tuples as conflict classes, the rate of false conflicts decreases which leads to less contention, hence better performance.

2.2.3 Optimistic Replication Protocols

In the optimistic approach to group-based database replication, transactions are immediately executed by the replicas to which they are submitted without any *a priori* coordination. Locally, transactions are synchronized according to the specific concurrency control



(a) PGR

(b) DBSM

Figure 2.4: Optimistic replication protocols

mechanism of the database engine.

Upon receiving the commit request, a successful transaction is not readily committed. Instead, its read set (RS) and its write set (WS) are collected and a termination protocol initiated. The goal of the termination protocol is to decide the order and the outcome of the transaction such that the global correctness criteria is satisfied. This is achieved by establishing a total order position for the transaction and certifying it (*i.e.*, checking for conflicts) against concurrently executed transactions. The certification of a transaction is done by evaluating the intersection of its RS (or WS in case of the snapshot-isolation criterion) with the WS of concurrent, previously ordered transactions. The formal definition and detailed explanation of the certification procedures can be found in [KA00, PGS03, WK05]. The fate of a transaction is therefore determined by the termination protocol and a transaction that would locally commit may end up aborting.

The two optimistic protocols considered, PGR and DBSM (Figure 2.4), ensure global serializability, but differ in their termination protocols. Both use the transaction's RS for the certification procedure. But, in PGR the transaction's RS is not propagated and thus only the replica executing the transaction is able to certify it. On the other hand, in the DBSM, the transaction's RS is propagated allowing each replica to autonomously certify the transaction.

In detail, upon the reception of the commit request for a transaction t, in PGR the

executing replica $TOCAST(m, g)$. Message m contains t's identification and t's WS and WV (recalling Section 2.2.1, WV means the values of the tuples in the WS). As soon as t is ordered, the executing replica certifies t and reliably multicasts the outcome to all replicas. The certification procedure consists on checking t's RS against the WS of all transactions committed locally since t's commit request. The executing replica then commits or aborts t locally and replies to the client. In the original protocol [KA00], a locking concurrency control mechanism was considered for the database engine which allowed to carry the certification process inside the database as part of the normal execution of the transaction. The RS was not extracted and was actually the read locks granted to the transaction. Upon the reception of the remote transaction's commit outcome each replica applies t's state changes through the execution of a *high priority* transaction consisting of updates, inserts and deletes according to t's previously multicasted WV. The high priority of the transaction means that it must be assured of acquiring all the required write locks, possibly aborting any locally executing transactions.

The termination protocol in the DBSM is significantly different and works as follows. Upon the reception of the commit request for a transaction t, the executing replica $TOCAST(m, g)$. Message m carries t's id, the version of the database on which t was executed, and t's RS, WS and WV. As soon as t is ordered, each replica is able to certify t on its own. The database version is a counter maintained by the database that is incremented every time a transaction commits.

For the certification procedure, in the DBSM each replica compares its database version with that of t: if they match t commits, otherwise t's RS are checked against the WS of all transactions committed locally since t's database version. If they do not intersect, t commits, otherwise t aborts. If t commits then its state changes are applied through the execution of a high priority transaction consisting of updates, inserts and deletes according to t's previously multicasted WV. Again, the high priority of the transaction means that it must be assured of acquiring all the required write locks, possibly aborting any locally executing transactions. The originating replica replies to the client at the end of the transaction.

Of particular relevance for the performance of these two protocols is the definition and representation of the transaction's read-set. Since it determines the outcome of a transaction certification it should reflect as accurate as possible which tuples a transaction actually reads. In the DBSM protocol, the read-set size may have a serious impact on the network bandwidth. In order to ease the network bandwidth usage not sending the entire read-set may be considered, but then serializability is compromised. Another approach is to upgrade the read-set granularity to table level, which leads to increased abort rate due to false conflicts. In the PGR protocol, these issues are avoided at the expense of an additional communication step.

When considering the snapshot-isolation correctness criterion, then both protocols

can be simplified and end up being the same. To satisfy snapshot-isolation, certification does not need to check Read-Write conflicts and thus the transactions' RS are not required. As such, the PGR protocol can be simplified by enabling a simpler Write-Write certification at all the replicas and eliminating the second communication step conveying the outcome of the transaction [WK05]. The DBSM protocol can also be simplified by not propagating the read-sets and using the simpler certification procedure.

In this family of replication protocols is interesting to assess the impact of network topology changes. Specifically, what is the impact on the system TPM (transaction per minute) when changing the properties of the communication network (LAN vs WAN). It is known that the probability of aborting a transaction in a optimistic concurrency protocol increases in a quadratic effect [Tho98] regarding its size. Therefore, this is also worth evaluating, specially the impact on wasted processing power due to execution of large transactions that end up aborting. Another interesting issue, is the impact of faults, whether in increasing transaction latency, due to network retransmissions of failed messages, or in processing throughput on heavily loaded replicas. Finally, it is also interesting to assess, in the DBSM protocol scope, the impact of defining the read-set granularity, either in terms of abort rate as in terms of network bandwidth usage.

2.3 Performance and Dependability Evaluation

Detailed evaluation of the performance and dependability of abstract protocols and of their implementations is required to examine each trade-off in different environments. Realistic tests are however costly to setup and run and depend on the availability of representative workloads and fault-loads. This is especially difficult when targeting large clusters or wide-area systems. Although often used, toy applications and micro-benchmarks are unable to disclose the subtle interactions with application semantics and dynamics (e.g. flow control issues and hot-spots) and with the environment (e.g. fault scenarios) as well as introduce significant probe effect.

System evaluation also depends on the availability of the complete target system. This precludes incremental development and early testing of individual components. Agile development methodologies have been attracting software engineers to focus on modeling and automated testing of compliance. Unit testing means that development starts by producing auxiliary test components directly from the model which are used, as the software product evolves, to ensure that it matches initial modeling.

2.3.1 Testing Environments and Benchmarks

Benchmarks are used to assess a system's performance and scalability. Stress-driven or specific context driven tests are conducted when running the benchmark. This allows

the detection of bottlenecks as well as assessing its validity under not so usual situations, which normally are disregarded during development. Benchmarking a system, involves deploying a suite of applications that setup the testing scenario according to the benchmark specifications. This usually implies deploying a fully working system. For instance, if one wants to use TPC-C [Cou01] or SPEC-Web [lSPEC05] benchmarks, a set of applications and a database needs to be installed. The resulting overhead of setting up the benchmark, either in terms of resources as in terms of time, may be prohibitive from the researcher point of view.

There are a number of tools to setup and control distributed tests and benchmarks, such as NetBed [WLS+02], ACME [ACM06], and TestZilla [Uni06], targeted specifically at performance. A large share of the complexity of such tools is directly related with the distributed nature of the system under study, namely, in performing consistent global observation of system state and properties while minimizing interference. Tools such as Facilita Forecast [Fac06] add to a distributed testing scenario the generation of representative loads, enabling load and stress testing to a wide range of applications. Distributed testing can also be performed in realistic testbeds for wide area networks such as Emulab [EMU] and PlanetLab [Pla] which provide transparent usage of the resources. They provide flexibility but the user still needs to deploy a real system to perform the test.

2.3.2 Simulation

Simulation is often defined as the technique to mimic behavioral interactions of a given environment, using a specified abstract model or instrument. The model or instrument are used to obtain reliable and valuable information on how the real system should evolve through time or even as a training playground [VOTC96]. Models may be categorized into physical or mathematical.

When modeling a system, one must take into account that the model should not represent the system *per se*. In fact, it should be a simplification of the real system. Nevertheless, the aspects under study should hold sufficient detail to draw valid conclusions about the real system [BINN00].

Simulation of computer systems uses the discrete event approach: Time advances as events are scheduled to happen. Therefore, if the system is just leaving instant in time t and the next event is scheduled to happen at instant t', the simulation time-line jumps from t to t' instantaneously. In such approach, components trigger events which are scheduled to happen in the future creating a dynamic simulation time flow. Components tend to be defined at a considerable level of abstraction, hence designing large scale simulation models is acceptable as well as their computation time. A nice example of how to use this simulation strategy is a CPU model. It may be modeled as a simple queue in which events are defined as the: the arrival of a job and the departing of a job. In detail,

upon a job, J_0, arrival, it is set to execute at the CPU. If a job, J_1, arrives in the mean time, it is put on hold. The service time is simulated by scheduling an event to happen at the end of the job execution. When J_0 execution reaches the end, the event is triggered, and job J_1 is set to execute in the CPU.

An implementation of a discrete event simulation approach is the Scalable Simulation Framework (SSF) [Cow99]. To build an SSF model, one identifies the objects of interest, *entities*, and their *attributes*. Entities may have their attributes changed over the time by *processes*, also named *activities*, which are triggered by the occurrence of *events*. Moreover, the system maintains a *state*, the collection of variables which the modeler specifies as being interesting for the study, describing the system at any time.

The SSF interface specifies additionally *incoming* and *outgoing channels* which act as event routes between entities and their associated processes. Delays are imposed on individual events, but also on channels and channel bindings, allowing the system to partition the simulation and take advantage of parallel processing [NL02, LNPP99, NL97]. The specification includes also the Domain Modeling Language (DML) [DML] that can be used to assemble models. Multiple implementations of the SSF interface are available in C++ [NL02, iSS] and Java [SSF]. The simple and clear interface of SSF provides means for intuitively and easily setting up simulation models. As an example, if one wants to simulate message passing in a network, one may bind channels between entities, acting as network cards, in a point-to-point manner. Events written into channels represent messages. Messages get delayed in the "network" accordingly to the delay specified at channel binding time or may be imposed specifically for each message when the event write operation takes place. These kind of setups may be configured using only a simple DML configuration.

A more detailed example is shown in Figure 2.5 and Figure 2.6. These compare real and simulated client/server interaction, respectively. Note that in these diagrams, and for the sake of comprehensibility, channels and entities are implicit, hence, asynchronous events suffixed with *Event* translate into write and reads into implicit channels. In Figure 2.5, the time it takes to execute a request is the sum of the partial latencies of calling, executing and receiving back the return value. The identical scenario is depicted in Figure 2.6, but this time using pure simulation. In this case, direct invocations are translated into events and simulation models are used instead of real implementations. This means that both, client and server, are now entities with associated processes. These are connected using channels from and on which events are read and written, respectively. As in the real implementation, the execution latency is determined by the time spent calling the method, executing the request and getting the return value. But now, invocation, execution, and reply latencies are modeled as event delays. Therefore, one may say that the simulation clock advances horizontally, each time an event operation (read/write to a channel is performed) and real time advances vertically as native instructions get executed.

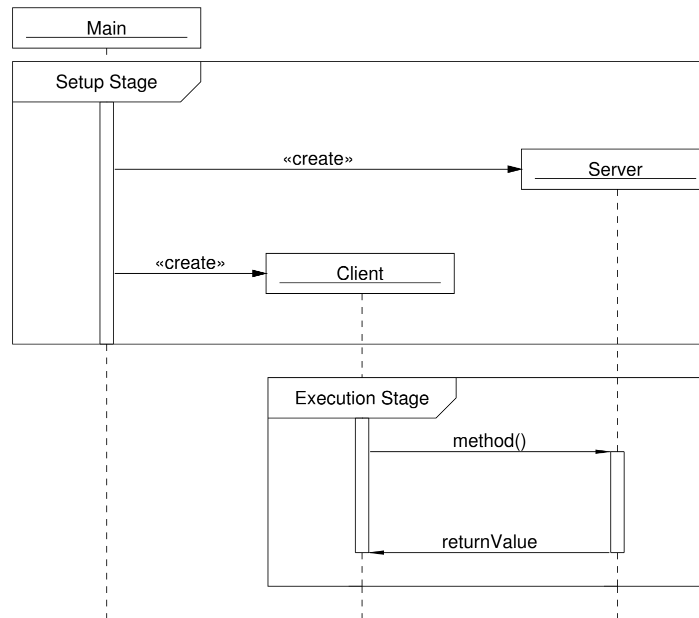


Figure 2.5: Real implementation: Client/Server interaction UML diagram.

A popular approach to evaluate designs, and specifically in the study of the dynamic properties of very large and complex systems, is the development of simulation models, making use of discrete-event strategy to operate simulation time-line. Namely, the development of the network infra-structure, protocols, and their applications is, most often, based on tools such as ns-2 [NS2] and SSFNet [CLL+99]. Although simulation models are usually distinct from final implementations, often using scripting languages to simplify modeling, it is sometimes possible to reuse library code by wrapping the functionality of simulated components in standard APIs, thus allowing functional testing.

Fine grained simulation of computer systems can also be used to create highly realistic although small scale testbeds. Namely, tools such as Simics and SimOS [RBDH97] simulate in detail computer systems allowing the execution of COTS binary-only operating systems and application software with unparalleled observability and lack of interference. On the other hand, the detail means that substantial computing resources are required to run realistic loads and that full implementations are required for testing. This can be improved by directly running operating system and application code in the host processor. This is the approach of UMLSim [Alm03] and FAUMachine [BS01], which additionally allow for fault injection for dependability evaluation. Nevertheless, full implementations are still required for testing.

2.3.3 Evaluation Framework

The ability to evaluate real implementations in a simulated environment has been proposed in CESIUM [AC97]. In this framework, implementations of communication pro-

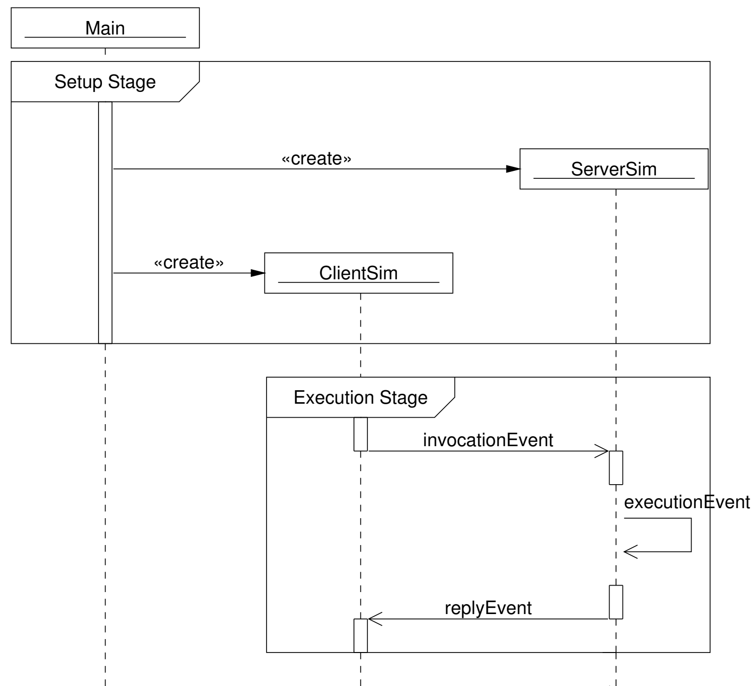


Figure 2.6: Simulation: Client/Server interaction UML diagram.

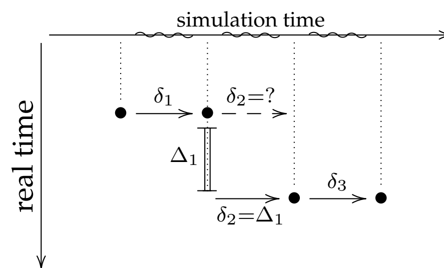


Figure 2.7: Executing simulated and real jobs.

tocols are tested for real-time properties. By running multiple instances of the implementation in a single address space within a discrete-event simulation model of the environment, centralized observation and manipulation of state is allowed with reduced interference. Scheduling and management of virtual-time by taking account of real time consumed by implementations, as obtained by profiling them, allows the system to be tuned to accurately reproduce real systems. This approach is thus the only that allows incremental substitution of model components by real implementations in performance evaluation.

In an event-driven simulation, time is incremented only by scheduling events with non-zero delays. The challenge when mixing real and simulated components is to ensure that the time actually spent executing real code is accurately reflected in simulation time. Thus this approach goes beyond the simple reuse of real code for simulation models and is able to reproduce timing properties of real systems [AC97].

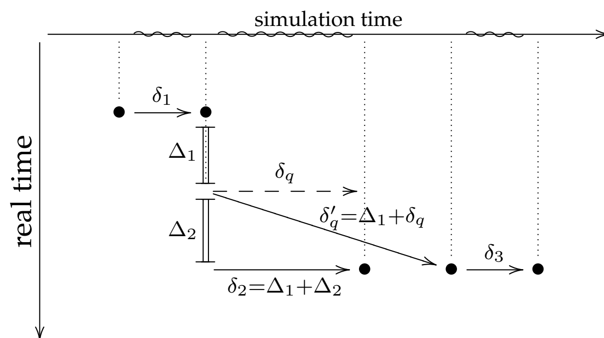


Figure 2.8: Scheduling events from real code.

In detail, this implies starting a profiling timer whenever real code is entered to account for native execution time. When execution re-enters simulation code, the profiling timer is stopped and the elapsed time used as an offset for all events scheduled. This is illustrated in Figure 2.7. From the simulation point of view, a job with duration δ can be set to execute at a specific instant t by scheduling a simulation event to enqueue it at simulated time t. Executing jobs with real code is layered on top of the same simulation mechanism. Figure 2.7, illustrates this with an example of how three queued jobs are executed. The second job is assumed to contain real code. The x-axis depicts simulated time and the y-axis depicts relevant real-time (*i.e.*, we ignore real-time consumed during execution of pure simulation code and thus pure simulation progresses horizontally). The x-axis shows also with an wiggly line when the simulated execution is taking place. Solid dots represent the execution of discrete simulation events. Scheduling of events is depicted as an arrow and execution of real code as a double line.

The first job in the queue is a simulated job with duration δ_1. After δ_1 has elapsed, execution proceeds to a real job. In contrast with a simulated job, one does not know beforehand which is the duration δ_2 to be assigned to this job. Instead, a profiling timer is started and the real code is run. When it terminates, the elapsed time Δ_1 is measured. Then $\delta_2 = \Delta_1$ is used to schedule a simulation event to proceed to the next job. This brings into the simulation time-line the elapsed time spent in a real computation. Finally the second simulated job is run with duration δ_3.

As a consequence of such setup, queueing (real code or simulated) jobs from simulated jobs poses no problem. Only when being run, they have to be recognized and treated accordingly. Problems arise only when real code needs to schedule simulation events, for instance, to enqueue jobs at a later time. Consider in Figure 2.8 a modification of the previous example in which the third job is queued by the real code with a delay δ_q. If real code is allowed to call directly into the simulation runtime two problems would occur:

- Current simulation time still doesn't account for Δ_1 and thus the event would be scheduled too early. Actually, if $\delta_q < \Delta_1$ the event would be scheduled in the

simulation past!

- The final elapsed real time would include the time spent in simulation code scheduling the event, thus introducing an arbitrary overhead in δ_2.

These problems can be avoided by stopping the real-time clock when re-entering the simulation runtime from real code and adding Δ_1 to δ_q to schedule the event with a delay δ'_q. The clock is restarted upon returning to real code and thus δ_2 is accurately computed as $\Delta_1 + \Delta_2$. In addition to safe scheduling of events from simulation code, which can be used to communicate with simulated network and application components, the same technique must be used to allow real code to read the current time and measure elapsed durations.

Chapter 3

Centralized Simulation

This chapter presents the proposed centralized simulation kernel and environment models that compose the distributed database evaluation framework. A description of how this framework is calibrated and validated according to a real system is also presented.

3.1 System Architecture

The system is comprised of four distinct modules: *i*) workload; *ii*) transactional database engine; *iii*) group-based replication protocols; and *iv*) network infra-structure. These are depicted in Figure 3.1 as a stack of components. Since the work presented is about database replication, each time an evaluation run is performed, the system contains several instances of the given stack, one for each database site.

On top of the stack one finds the workload generator. It mimics clients issuing transaction requests to the database engine. Transactions submitted to the database engine are executed and if any item is updated during execution, then the update transactions cannot commit before the replication protocol coordinates with the other replicas. Therefore, the database engine interacts with the replication protocol in order to propagate the updates. Once the replication protocol propagates the updates to all other replicas using the network infra-structure, it then warns the database engine that it has terminated and the transaction is ready to be committed or even aborted. Read-only transactions do not need synchronization between replicas, therefore they can commit without any interaction with the replication protocol module.

Detailing the stack presented, one finds that the system under study are the replication protocols, therefore real implementations are used in the replication part of the stack. Database and network are abstracted as simulation models. The workload generator is actually half real, half abstraction, because at its core lies the same collection of distributions used to generate transactions either for a real database as for the abstrac-

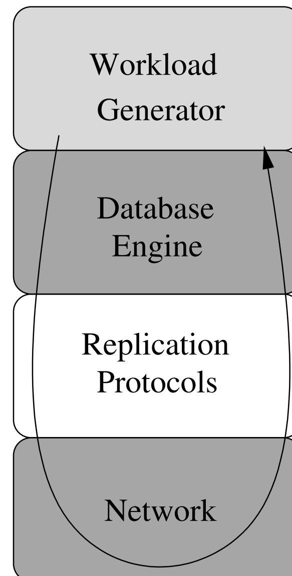


Figure 3.1: Simple overview of the system architecture.

tion presented in the model. In order to connect the generator to the simulated database engine a simple adapter is needed to translate real transactions requests into simulated transactions. In Figure 3.1, white background boxes represent real implementations while dark background boxes represent simulation abstractions. The light shaded background box represents the workload generator. Additionally, the arrow represents an update transaction execution flow.

Using a realistic traffic generator submitting transactions to the system under test is most valuable, since it enables realistic testing of the database engine, the replication protocols and the network. It also allows the system under test to face real world scenarios in which several minor details always arise when deploying a system that has only been tested with toy applications.

The next sections detail the architecture, using a top-down approach according to the picture presented.

3.2 Simulation Model

3.2.1 Workload Model

The workload model is defined by the TPC-C [Cou01] benchmark. It is the industry standard on-line transaction processing (OLTP) benchmark which mimics a wholesale supplier with a number of geographically distributed sales districts and associated warehouses. The database contains the following tables: *(i) warehouse*; *(ii) district*; *(iii) customer*; *(iv) stock*; *(v) orders*; *(vi) order line*; *(vii) history*; *(viii) new order*; and *(ix) item*. The

Transaction	Probability	Description	Read-only?
Neworder	44%	Adds a new order into the system.	No
Payment	44%	Updates the customer's balance, district and warehouse statistics.	No
Orderstatus	4%	Returns a given customer's latest order.	Yes
Delivery	4%	Records the delivery of orders.	No
Stocklevel	4%	Determines the number of recently sold items that have a stock level below a specified threshold.	Yes

Table 3.1: Transaction types in TPC-C.

traffic is a mixture of read-only and update intensive transactions. A client can request five different transaction types as follows: Neworder, adding a new order to the system (with 44% probability of occurrence); Payment, updating customer's balance, district and warehouse statistics (44%); Orderstatus, returning a given customer latest order (4%); Delivery, recording the delivery of products (4%); Stocklevel, determining the number of recently sold items that have a stock level below a specified threshold (4%). TPC-C table related information and transaction pattern are best depicted in Table 3.1 and Table 3.2. The model accurately follows database scaling rules defined by TPC-C and the database is scaled according to the number of clients. Namely, an additional warehouse should be configured for each additional 10 clients and initial sizes of tables are also dependent on the number of configured clients.

Each client is attached to a database server and produces a stream of transaction requests. When a client issues a request it blocks until the server replies, thus modeling a single threaded client process. After receiving a reply, the client is then paused for some amount of time (think-time) before issuing the next transaction request.

The model abstracts away benchmark requirements such as screen load and background execution, which are not relevant for the work presented here. In fact, these are not significant for the evaluation of replication and group communication protocols, therefore, TPC-C is used as the basis for a realistic application scenario and not as a benchmark.

During a simulation run, clients log the time at which a transaction is submitted, the time at which it terminates, the outcome (either abort or commit) and a transaction identifier. The latency, throughput and abort rate of the server can then be computed for one or multiple users, and for all or just a subclass of the transactions.

	Number of items			
Relations	100 (Cli.)	2000 (Cli.)	4000 (Cli.)	**Tuple size**
Warehouse	1×10^1	2×10^2	4×10^2	89 bytes
District	1×10^2	2×10^3	4×10^3	95 bytes
Customer	3×10^5	6×10^6	12×10^6	655 bytes
History	3×10^5	6×10^6	12×10^6	46 bytes
Order	3×10^5	6×10^6	12×10^6	24 bytes
New Order	9×10^4	18×10^5	36×10^5	8 bytes
Order Line	3×10^6	6×10^7	12×10^7	54 bytes
Stock	1×10^6	2×10^7	4×10^7	306 bytes
Item	1×10^5	1×10^5	1×10^5	82 bytes
Total	$\approx 5.1 \times 10^6$	$\approx 10 \times 10^7$	$\approx 2 \times 10^8$	

Table 3.2: Size of tables in TPC-C.

3.2.2 Transaction Processing Model

The database server model, named *SSFDb*, handles multiple clients and is modeled as a scheduler and a collection of resources, such as storage, cache, processing units and a concurrency control policy [ACL87, ACL85]. Each transaction is modeled as a sequence of operations, which can be one of: *i*) fetch a data item; *ii*) do some processing; *iii*) write back a data item; *iv*) call replication engine; and *v*) issue a lock related operation. Upon receiving a transaction request each operation is scheduled to execute on the corresponding resource. As an example, the generic replicated transaction execution path, is shown in Figure 3.2. Every resource is modeled as a simple queue. For example, I/O requests arriving at storage will have to wait if the I/O throughput is already at its maximum usage. The same happens in the CPU pool, if a job arrives and no CPU is free, it will have to be put on a waiting queue or preempt, if that is the case, the execution of an executing job.

Processing operations are scaled according to the configured CPU speed. Each is then executed in a round-robin fashion by any of the configured CPUs. Additionally, simulated CPUs are also used to schedule real jobs by the centralized simulation runtime. Therefore, transaction execution can be preempted to assign the CPU to real jobs, enabling concurrent execution between real and simulated jobs.

I/O operations are modeled using a storage abstraction that specifies procedures for fetching and storing items. Its service rate (throughput) is defined by the number of allowed concurrent I/O requests and the latency of a single request. Each request manipulates a write attempt, meaning that the cache hit ratio is very close to 100%, hence storage bandwidth becomes configured indirectly.

Operations for fetching and storing items are submitted to the concurrency control module (lock manager). Depending on the policy being used, the execution of a transac-

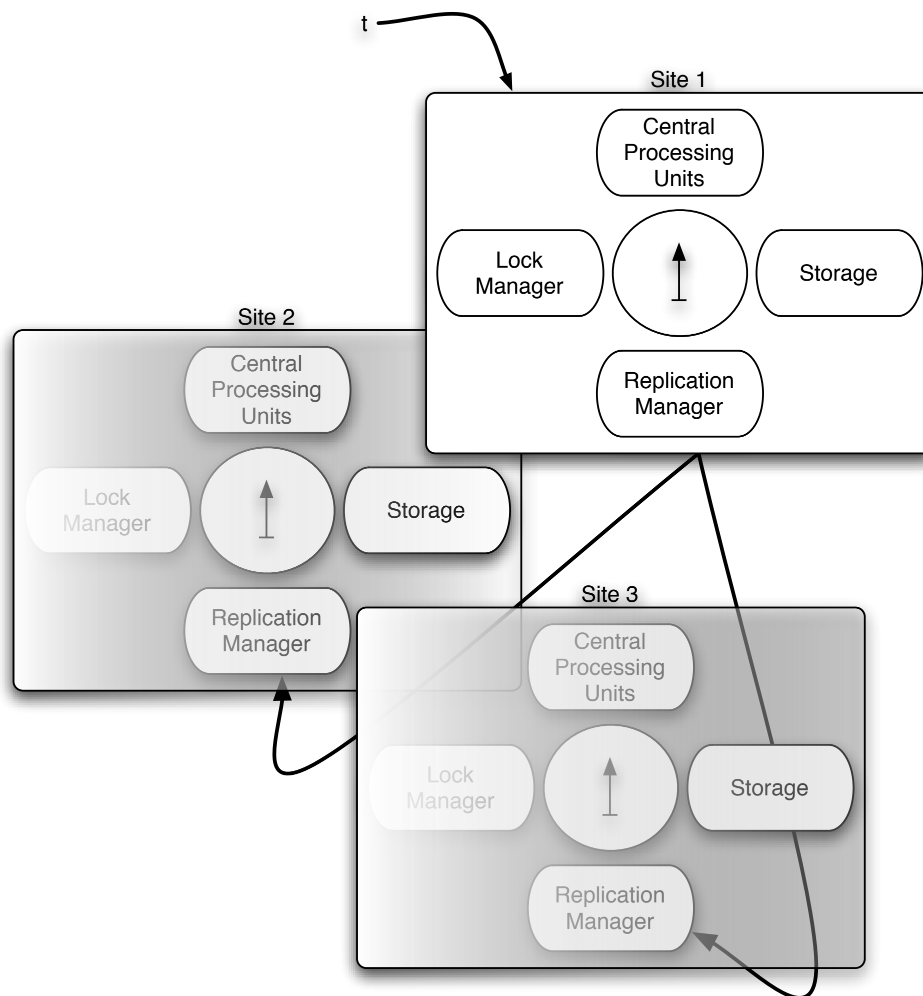


Figure 3.2: Simulated transactions and replicas interaction.

tion can be blocked between operations. Items get locked accordingly to the user defined locking policy. The concurrency control may be based on timestamp or strict two phase locking (2PL) protocols [BHG87]. In a timestamp locking policy, items fetched for reading are ignored, while items updated are exclusively locked. When a transaction commits, all other transactions waiting on the same locks are aborted due to Write-Write conflicts. In the strict two phase locking all items get locked. Whenever a transaction aborts, its locks are released and can be acquired by any subsequent transaction. In addition, all locks are atomically acquired, and atomically released when the transaction commits or aborts, thus avoiding the need to simulate deadlock detection. This is possible as all items accessed by the transaction are known beforehand.

The transaction processing model also offers hooks to plug in replication protocols. This is actually one of the main interfaces between the simulated environment and the system under test (*i.e.*, replication protocols). In detail, these offer the possibility to: *i*) intercept queries upon arrival, to allow classification procedure and ordering required by the conservative protocols; *ii*) intercept read and write-sets upon entering commit

and producing realistic amount of write values which are to be transmitted over the network, hence will produce real bandwidth consumption; *iii*) handle installation of remote updates, for which there is no directly attached client entity. Note that the interaction between simulation and real code is also performed at this level by using the centralized simulation kernel (see Section 3.1).

During a simulation run, the usage and length of queues for each resource are collected and used to examine in detail the status of the server. This is most useful to find contention in the system. For example if the queue at one resource becomes extensive then a bottleneck has been found. Then, one may confirm the bottleneck hint by examining the usage data to verify which is the optimal throughput rate of the resource.

3.2.3 System Under Test

The system under test is composed of actual implementations of database replication protocols an group communication. Three different implementations are used as a case study in this work. Namely, Database State Machine (DBSM), Postgres-R (PGR) and Conservative (CONS) replication protocols.

When considering the conservative protocol, transactions are set to classify prior to their execution. This involves a call to the replication layer in order to classify the transaction. While the classification is not concluded, the transaction is put on-hold. Once it finishes, the transaction may begin its execution. By the time the commit operation is issued, the write values are gathered and sent to the other replicas, hence another call to the replication service is issued. On the other hand, if the DBSM protocol is considered, there is only one call to the replication module. It happens when the transaction tries to commit. This is also the case for PGR, which requires an additional communication step when compared to the DBSM. All in all, every update transaction goes through the replication layer at least once. Therefore, there is one or more interactions between real code prototypes and simulation.

Each of the replication protocols rely on an atomic multicast protocol, which is provided by the group communication layer. Such primitive is implemented in two stages. A view synchronous multicast protocol and a total order protocol. The former, view-synchronous multicast, works in two phases. First, messages are disseminated, taking advantage of IP multicast in local area networks and falling back to unicast in wide-area networks. Then, reliability is ensured by a window-based receiver initiated mechanism similar to TCP/IP [PTK94] and a scalable stability detection protocol [Guo98]. Flow control is performed by a combination of a rate-based mechanism during the first phase and the window-based mechanism during the second phase. View synchrony uses a consensus protocol [SS93] and imposes a negligible overhead during stable operation.

The goal of the stability detection protocol is to determine which messages have al-

ready been delivered to all participants and can be discarded from buffers. It is therefore a key element in the performance of reliable multicast. Stability detection works in asynchronous rounds by gossiping: *i*) a vector S of sequence numbers of known stable messages; *ii*) a set W of processes that have voted in the current round; and *iii*) a vector M of sequence numbers of messages already received by processes that have voted in the current round. Each process updates this information by adding its vote to W and ensuring that M includes only messages that have already been received. When W includes all operational processes, S can be updated with M, which now contains sequence numbers of messages discovered to be stable.

Total order is obtained with a fixed sequencer protocol [BvR94, KT91]. In detail, one of the sites issues sequence numbers for messages. Other sites buffer and deliver messages according to the sequence numbers. View synchrony ensures that a single sequencer site is easily chosen and replaced when it fails. By implementing total order within the prototype, it becomes possible to later explore several optimizations of atomic multicast in the context of transaction processing. Namely, semantic reliability [PRO03] and optimistic total order [PS03, SPMO02]. Semantic reliability improves throughput stability in heterogeneous and wide area networks by discarding messages that become obsolete while still in transit, for instance, because a transaction is known to have aborted. Optimistic total order recognizes that it is possible to exploit the spontaneous order of messages which happens with high probability to optimistically start processing transactions. If the order turns out to be wrong, the execution is rolled back and the transaction is redone in the correct order.

3.2.4 Network Model

The network model is defined by using the components from a library of components that can be reused. This is the case of the SSFNet [CLL+99] framework, which models network components (e.g. network interface cards and links), operating system components (e.g. protocol stacks), and applications (e.g. traffic generators). Complex network models can be configured using such components, mimicking existing networks or exploring particularly large or interesting topologies. The SSFNet framework provides also extensive facilities to log events. Namely, traffic can be captured in the same format used in real networks and thus the log files can be examined using a variety of existing tools.

The interface between the group communication and the network is the other main interface between the system under test and the simulated environment. It offers a simplified version of the standard socket API for connectionless protocols. Again, interaction between simulation and real code is performed at this level by using the centralized simulation kernel.

3.3 Simulation Kernel

In the previous section prototypes of the replication and group communication protocols were said to be embedded within simulation. This section explains how this is achieved by detailing the simulation kernel and how it handles real time execution. It starts by comparing a real system and a simulation model. Then it goes on demonstrating how to mix both using the simulation kernel developed.

3.3.1 Real-Time

Running selected components with simulation models of realistic environments, workloads, and fault-loads is accomplished using a centralized simulation kernel [AC97]. This approach is implemented as a small extension to the Scalable Simulation Framework (SSF) [Cow99] specification that greatly simplifies interfacing real implementations within simulation models while accurately reproducing the timing behavior of real systems.

The extension of the SSF interface is very simple and consists on being able to designate selected entities as *real-time entities* by overriding the isRealTime() method. A process associated with such entity becomes a *real-time process* and behaves as follows: the profiling clock is started when the process reads an event from an incoming channel and stopped whenever the process writes an event to an outgoing channel. Code executed between writing an event and reading another event is therefore not accounted for. After stopping the profiling clock, the real-time process is not rescheduled until simulation time has advanced by as much time as real execution took. The event is actually written to the channel only after simulation time has catch-up. The proposed programming interface was implemented in Java as the *MinhaSSF* package. This was required as source of the existing implementation is not available.

Accounting real-time within the simulation kernel is easily implemented in existing SSF implementation as this boils down to taking into consideration real time processes whenever entering the simulation runtime and minor changes to the scheduler. A key issue is the profiling clock used to measure real-time. It is important that the method used allows a fine-grained measurement of time by one operating system thread even if other concurrent threads are running and can thus preempt the desired thread. This is achieved in the Linux operating system using the perfctr patch [Pet04], which offers a virtualized hardware cycle counter for each operating system process.

The proposed semantics is targeted at client/server interactions between simulated and real components. Real code calling into simulation by means of writing an event will suspend accounting of real time, which is resumed upon reading the return event. Conversely, when behaving as a server, accounting of real time starts upon receiving an invocation event from simulated components and until a return is written.

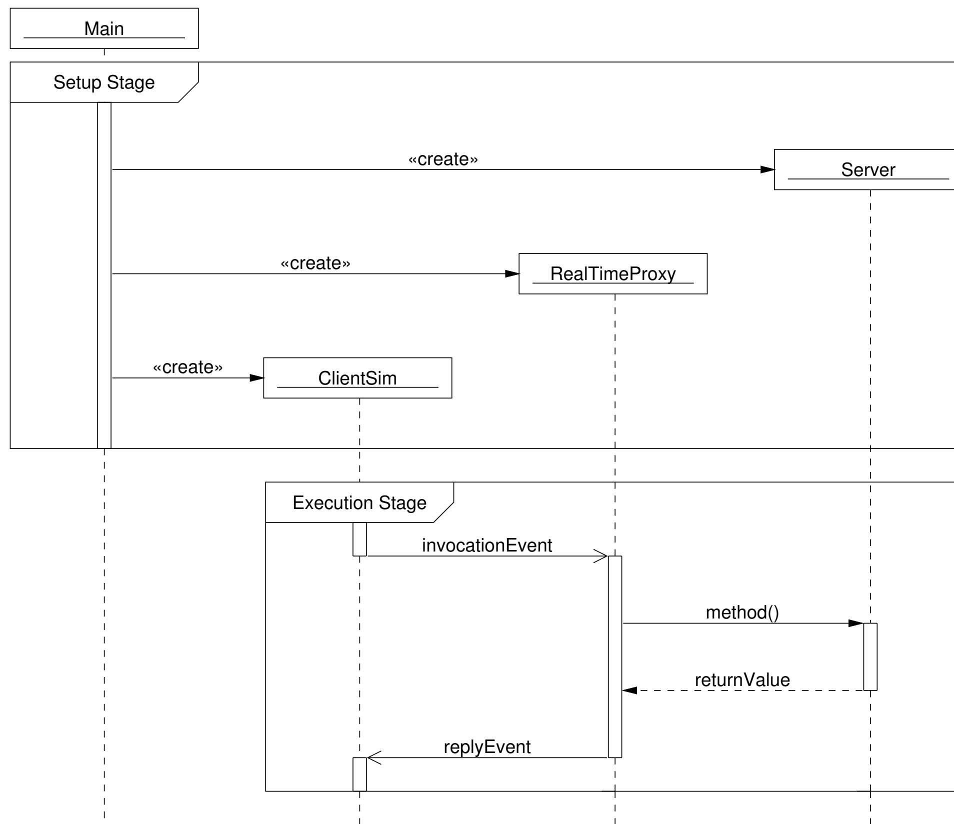


Figure 3.3: Simulated client calls server real implementation.

The SSF specification as well as the extensions mentioned in this section are implemented in Java and available to the GORDA project[Con04] and are to be release in the near future as open source.

3.3.2 Client/Server Utility Classes

To avoid the tedious and error prone task of manually writing stubs for every interaction, *i.e.*, which translate events into invocations and back, a set of proxy classes has been implemented. These are realized using Java reflection and provide a simple and clear interface that enable generalized usage and complete isolation between simulation models and real implementation. A proxy, named *SimulationProxy*, is responsible to transform an invocation made by real code into a pair of events (write and read operations), that transparently interface with simulation code. The second proxy, name *RealTimeProxy*, does the reverse operation, listening for events, invoking real code and replying the result.

The role of dynamic proxies is better understood with an example. Recalling the previous client/server example from Section 2.3.2, one is able to easily reuse the real server implementation in conjunction with the simulated client, as Figure 3.3 illustrates. Keep in mind that entities and channels are implicit in the diagrams, as already previously stated. The server implementation is embedded into a RealTimeProxy, which in its turn

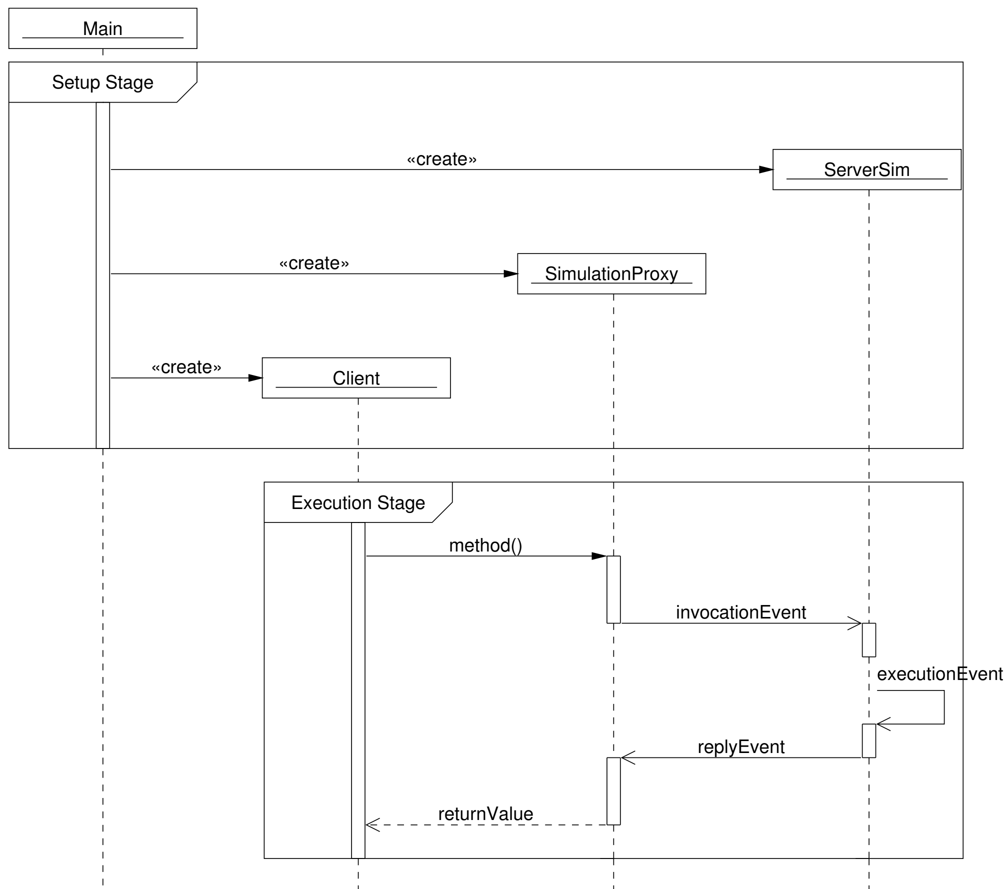


Figure 3.4: Client real implementation calls simulated server).

is a MinhaSSF real-time entity. This proxy contains a process that reads incoming events from an invocation channel and translates them into calls to the server implementation. At the same instant the process reads the invocation event, the simulation kernel implicitly starts a timer which keeps track of time spent in the real invocation, which is the next thing the proxy performs after reading the event. When the method returns, the proxy writes a reply event containing the result into the reply channel and the kernel implicitly suspends the timer. The reply event is delivered to the simulated client, after the time in the real execution elapses in the simulation clock. In the mean time, the RealTimeProxy process is not scheduled again until the simulation clock catches-up.

Calling simulation from real code is also possible due to the other proxy: *Simulation-Proxy*. Taking the client/server example, one more time, and now considering the client as the real implementation and the server as a simulation model, its usage is depicted in Figure 3.4. The SimulationProxy implements the interface of the server, exposes it to the other objects and provides channel end-points that can be connected to the simulated server to convey invocations and replies. A reference to the proxy object is provided to the real Client instance making it believe it has actually a reference to a server implementation. Once the client calls the *method()*, the proxy translates this call into an invocation

Resource	Properties
Processor	2×2.4 GHz AMD Opteron
Storage	1 dedicated 100Gb partition
RAM	4 GBytes
Operating System	Linux 2.6.10-1.737 (Fedora Kernel)
Filesystem	ext3
Database Management System	PostgreSQL 8.0

Table 3.3: System specifications.

event (*invocationEvent*), writes it to the channel, and blocks waiting for the reply. At this moment the MinhaSSF kernel suspends accounting of real time associated with the real client execution. When the server receives the event, which only happens after the simulation clocks elapses the time accounted in the client execution, it simulates the execution with an event (*executionEvent*) and sends back the result by writing an event (*replyEvent*) into the reply channel. Upon delivery of the reply event, the proxy collects the result and handles it back to the client as the return value of the fake *method()*, so control is back in the client implementation. At the same time the timer is resumed, hence the kernel restarts accounting execution time of client instructions.

3.4 Model Calibration

This section presents how the simulation model is calibrated. In detail, how to configure simulation components so they can reproduce accurately a real environment. The purpose of the calibration is to *tune* the model in order to guarantee generalized conclusions.

The calibration procedure is driven by the available hardware for testing, hence the model is configured accordingly. The hardware consists in one HP Proliant dual Opteron processors machine. It has 4 GBytes of RAM memory and uses a fibre-channel attached storage with 1 TByte in a RAID-5 configuration. The operating system used is Linux, kernel 2.6.10-1.737, from Fedora, and the database engine used is PostgreSQL v8.0, having all data stored in a dedicated partition sized in 100Gb, hence no other processes perform I/O into that device. Emulated TPC-C clients are configured to execute from remote machines and communicate with the DBMS over a LAN.

Database calibration is performed using the described hardware and running a benchmark with only one emulated TPC-C client. The logs collected are used to tune the simulation model. It is adjusted in two distinct levels: *i*) it needs to be setup according the hardware specifications of the real system (the same number of CPUs, the same storage throughput, ...); and *ii*) resource consumption must follow the same profile as in the real system (*i.e.*, the CPU must be under the an equivalent load, the number of access to storage must be equivalent, ...). Using the calibration setup, TPC-C runs with 10 clients, in both the real and simulated systems, are compared to verify if they match.

Transaction Name	Empirical Distribution	Estimators	
Delivery	normal	mean=143698975.769	sd=2332369.0642
Neworder	uniform	min=6450113.77352	max=16829661.7371
Orderstatus	normal	mean=1658057.33333	sd=830521.190802
Stocklevel	uniform	min=1845659.43141	max=2328872.56859
Payment	normal	mean=2261806.48101	sd=213283.617067

Table 3.4: CPU Times distributions (nanoseconds).

3.4.1 CPU

The CPU abstraction is modeled as queue in which the arrival process and the service time distribution follow an exponential distribution. The service time for each job is configured according to the transaction type. Given this, execution times are determined by measuring the time a transaction actually spends in a real CPU in a real execution. Therefore, simulated transactions spend equivalent times to the ones that a real transaction would spend in the real system.

In order to obtain CPU processing times, transactions need to be profiled. To accurately obtain the amount of CPU time consumed by each transaction, an instrumented version of PostgreSQL was developed. In PostgreSQL, a process, named the *backend*, handles a single transaction execution from start to end. This makes it easy to use the Linux perfctr patch [Pet04] to obtain the time spent processing by reading the virtualized per-process cycle counter.

In detail, a CPU timestamp counter is used, which provides accurate measure of elapsed clock cycles. By using a virtualization of the counter for each process, measurements of process virtual time (*i.e.* the time elapsed when the process is not scheduled to run is not accounted for) are also obtained. To minimize the influence in the results, the elapsed times are transmitted over the network only after the end of each query (and thus out of the measured interval), along with the text of the query itself. The time consumed by the transaction's execution is then computed from the logs. By examining the query itself, each transaction is classified.

The analysis conducted using the logs of the real execution. The initial 17 minutes of the run (100 transaction samples) and the aborted transactions are discarded, so warm-up effects are minimal, and the percentile 90% of the remaining, is considered. The 90% percentile eliminates deviating samples which are due to operating system overhead (swapping and process scheduling) as well as other system daemons interference. The resulting histograms (Appendix A) provide a hint on the empirical distribution that the collected samples follow. The profiling process also allowed to fine tune the PostgreSQL configuration as well as optimize the TPC-C implementation.

Once the logs become fully analyzed and the histograms plotted one may intuitively

find an empirical statistical distribution describing the processing times required for each transaction type (Table 3.4). Therefore, using the *fitdistr* function from *MASS* package of the R [ea97, Dal02] project for statistical computation, one is able to fit the empirical distribution to a theoretical statistical distribution. Details of the fitting process are presented in Appendix A.

3.4.2 Storage

The storage calibration is accomplished by considering the number of items that a transaction accesses for writing. Again, like in the CPU time profiling, the PostgreSQL engine had to be modified so the number of items a transaction tries to update gets logged. This is achieved by creating update, delete and insert triggers that keep track of the items accessed during the transaction execution. Upon a commit request, this information is passed to the logger which flushes the information to a file and frees memory kept for holding this information.

There are two situations to take into account: *i*) fetching; and *ii*) writing an item. The latency of reading operations is minimal judging from the results obtained, in the tests performed. In fact, despite the warm-up period, the cache hit ratio tends to 1.0, meaning that zero time is spent in I/O for reading. Writing operations requires studying the writes performed in the database log file and the writes performed in the database data space. Writing to the log is critical because it implies at least one disk flush when a transaction issues the commit instruction. This has impact in the total transaction execution latency. It has been verified that, writing to the data storage space does not produce significant impact in the final latency. Explaining this behavior is the fact that PostgreSQL keeps a background writer process that is in charge of flushing the dirty data to the database data space. This is performed asynchronously and most often outside transaction execution context, thence it does not count for the total execution latency.

$$Y = m \cdot X + b \tag{3.1}$$

$$m = \frac{Y_2 - Y_1}{X_2 - X_1} \tag{3.2}$$

In order to calculate the storage service time, the time spent on each log write operation is measured. But knowing the latency of a log write operation is not enough, one needs to find the correlation between the number of log writes and the number of items accessed. This need is imposed because the input for the simulated storage is the number of items a TPC-C transaction writes and not the number of log writes. With this information the storage model is configured using the writes-items relation, the latency of each access and a concurrent parameter that states how many concurrent accesses may

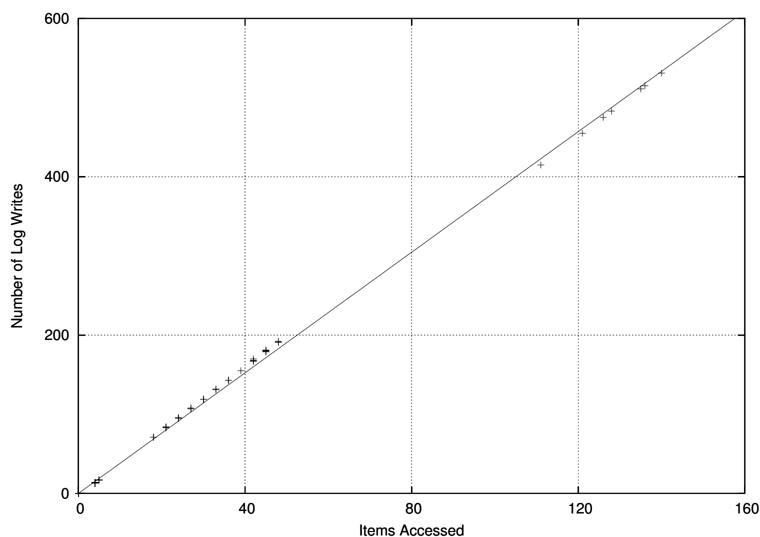


Figure 3.5: *XLogWrite* calls as a function of the number of items accessed.

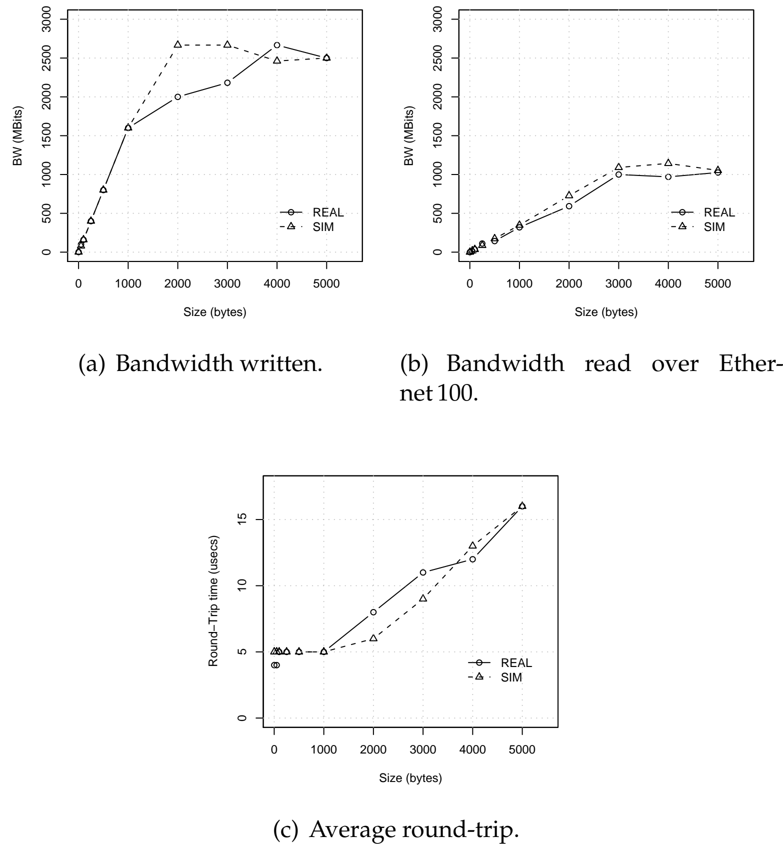
Parameter	Value	Short Description
m	3.8088	Slope of the ratio line between items and log writes.
logAccessDelay	986 ms	Average *XLogWrite* call latency.
parallelism	8	Parallel log writes.

Table 3.5: Storage simulation model parameters.

be performed. In order to obtain the number of log writes performed, one needs to export the number of calls to the PostgreSQL *XLogWrite* function and measure its time. By plotting, for each transaction, the number of items accessed *versus* the number of calls to *XLogWrite* one is able to find a linear relation between both. Figure 3.5, shows the number of log writes as a function of the items accessed. It is clear that there is a linear relation between both. The $2D$ Cartesian coordinate system, states that a line is described by Equation (3.1). Since b is where the line intersects the ordinate axis when X is equal to zero, then its value is obviously zero (Figure 3.5). The line slope is determined by the expression given by Equation (3.2). Therefore, if one considers the two given points $X_2 = (140, 531)$ and $X_1 = (13, 4)$ one is able to determine the line slope without any hassle: $m = \frac{531-13}{140-4} \approx 3.8088$. In addition, Figure 3.5, also depicts, as a solid line, the extrapolated equation: $Y = 3.8088 \times X$.

The number of parallel log writes is determined by the average number of calls between two consecutive flush operations. Flush operations force disk I/O. The measured number of log writes without any flush operation was 8.

Table 3.5, presents the storage simulation model final parameters obtained from the calibration process.



(a) Bandwidth written. (b) Bandwidth read over Ethernet 100.

(c) Average round-trip.

Figure 3.6: Validation of the centralized simulation runtime.

3.5 Validation

Simulation models are a simplification of the real system they mimic. The model presented is no different, therefore the match between the real system and the abstraction is by no means an absolute match. Nonetheless, a validation procedure is presented in this section, in order to stress the fact that the developed model performs quite similar to the real system.

Network model and its configuration are validated by comparing the resulting performance measurements of the model to those of the real system running the same benchmark.

Figure 3.6(a) shows the maximum bandwidth that can be written to an UDP socket by a single process in the test system with various message sizes. Figure 3.6(b) shows the result of the same benchmark at the receiver, limited by the network bandwidth. Finally, Figure 3.6(c) shows the result of a round-trip benchmark. The difference observed with packets with size greater than 1000 bytes is due to SSFNet not enforcing the Ethernet MTU in UDP/IP traffic. Deviations from the real system are avoided by restricting the

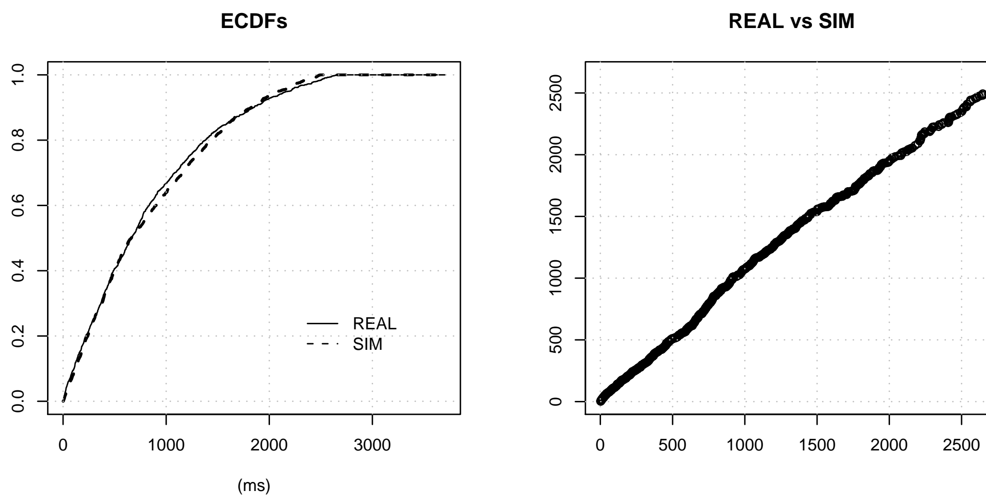
Transaction	Average TPM		Average Latency (ms)	
	Simulation	Real	Simulation	Real
Delivery	2.64	3.66	151.3	192.2
Neworder	33.37	32.43	12.82	11.57
Payment	29.88	30.14	3.21	3.23
Orderstatus	3.15	3.09	1.63	1.21
Stocklevel	2.87	3.18	2.08	2.63
TOTAL	71.62	72.93	7.328	6.97

Table 3.6: Simulation vs Real: TPM and latency results.

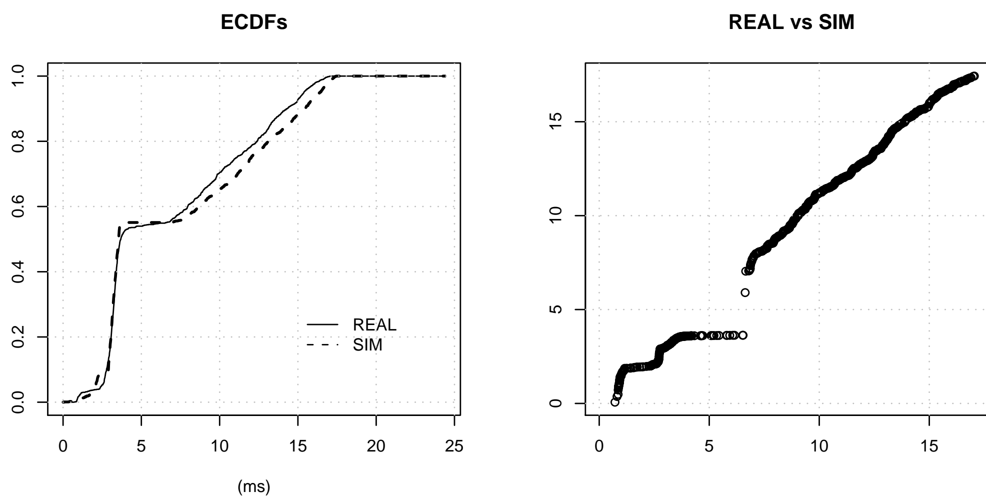
size of packets used to a safe value.

The last stage of the validation process is to determine how close to the real system the simulation models performs. Taking the configuration values obtained from the storage and CPU calibration, one is able to perform a simple run on the real system and compare the outcome with the same run on the simulation model. The metrics used for comparison are: transaction inter-arrival and execution latency. The former indicates the throughput, while the latter shows how much time a transaction spends inside the database engine during its execution. The abort rate is not taken into consideration, because is its practically zero in both runs.

The validation scenario considers ten emulated TPC-C clients issuing transactions requests to the database. Figure 3.7(b) and Figure 3.7(a), present the comparison of the latency and inter-arrival of transactions considering the percentile 90%. Note, that these plots show the Empirical CDF of the two runs and present a Q-Q plot to determine how close they are to each other. In a Q-Q plot, a 45° degrees line states that both plots are perfectly matching. Taking the Q-Q plot into consideration, both are very much alike either in terms of inter-arrival as in terms of latency, despite the deviation for a small amount of samples in the latter. This deviation is not significant due to the small amount of samples. Table 3.6, depicts numerical values for the average TPM and latency. One may find that real and simulated runs perform real close, hence the system becomes calibrated and validated. For additional details (per transaction analysis), please refer to Appendix B.



(a) Inter-arrival.



(b) Latency.

Figure 3.7: Simulation vs Real comparison.

Chapter 4

Case Study

This chapter presents a case study featuring the replication protocols put forward in Chapter 2. Special attention is payed to the performance evaluation of the replication system, since these measures strongly define the applicability of the approach. Evaluation was performed in ideal network environments as well as in faulty networks. The simulation environment was based on the centralized simulation environment, described in Chapter 3.

4.1 Motivation

The trade-offs implicit in the design of each protocol described in Chapter 2 can now be evaluated within a common framework and in a variety of environments. In detail, one wants first to answer questions such as:

- Is the replication protocol scalable, *i.e.*, is the system capable of operating under heavy loads, and in a large cluster?
- How does performance degrade when going from a local area network into wide area network, specially regarding the increase on latency for message transmission?
- What is the impact on performance if messages get lost in the network, and which transactions will suffer most in the presence of faults?
- How differently does the system perform when comparing Write/Write (Snapshot-Isolation) and Read/Write consistency criteria?

When considering optimistic replication protocols, one is also interested on the following:

- Does network become a bottleneck when sending read sets between replicas and if so, is there a way to reduce its size? What is the impact on the certification process when compressing the read set?
- Which transactions end up aborting most frequently and why?
- Do large transactions end up aborting most due to their longer execution latency?
- Does PGR benefits from not sending the read set over the network at the cost of an additional communication step?

On the other hand, when considering conservative replication protocols one wants to know the following:

- How should one consider the traffic profile to determine conflict classes?
- What is the impact on performance when defining coarse grained conflict classes (e.g., table level conflict classes)?
- How does contention in conflict class queues increase transaction latency?
- Is transaction latency directly proportional to contention due to class-level conflicts?

Answering these requires building large scale testing scenarios, changing the behavior of the database management system model, injecting faults and finally a number of different observations of external and internal metrics, which would be hard to accomplish in a real setting.

4.2 Scenarios

Given the motivations in the previous section, a case study allowing the answer of such questions is presented in this chapter. It aims at providing an extensive empirical observation of how the replication protocols introduced in Chapter 2, behave under different environments. The selected application model is based on one of the TPC set of benchmarks, and has already been exhaustively described in the Section 3.2. Replication is achieved having nine sites with the exact same instance of the TPC-C database. Hosting machines are modelled as HP Proliants with two AMD Opteron(tm) Processor 250 processors with 4GB of RAM. For storage, a fibre-channel attached box with 4×36GB SCSI disks in a RAID-5 configuration is considered. Details on how the simulation model calibration is accomplished may be found in Section 3.4.

Each site handles transactions submitted by clients in a multi-master way. The load is symmetric, *i.e.*, every site has the same number of clients connected to it, and every client

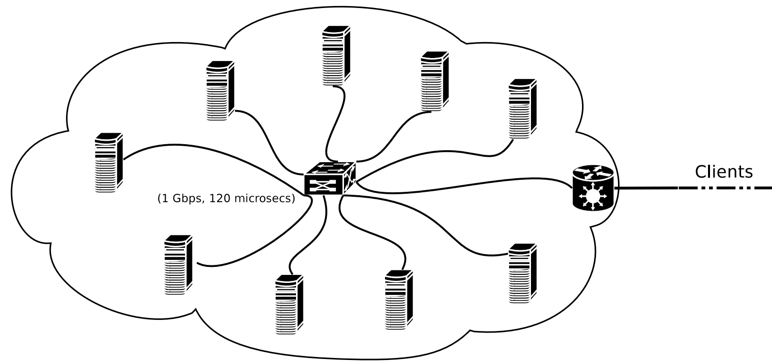


Figure 4.1: Local area network configuration.

is modelled as a sequential process. If a client tries to update the database, issuing an update transaction, it must be replicated to the other replicas, hence an update transaction submitted at site S_0 must also update every other site in the system. For the sake of clearness, transactions submitted by a client at a site will be referred to as *local transactions* and transactions submitted at any site by the replication protocols (remote updates) will be referred to as *remote transactions*.

As for the communication infra-structure, two different approaches are considered. The first approach considers nine sites communicating using low-latency and high bandwidth network channels, similar to a cluster environment. All sites are in the same network, hence they all operated in a LAN infra-structure. Links considered have 1 Gbps of bandwidth and the transmission latency is around 120 μs. Figure 4.1, depicts this network configuration.

In the second approach, sites are spread over a wide area network. The original nine sites are grouped on clusters of three, and each of these clusters exhibit the same properties of the LAN network mentioned previously. Intercommunication between groups is accomplished through a wide area network. Messages sent from one cluster to another, go through a border router which communicates with a backbone router. The communication channels between each border router and the backbone router are configured with 100 Mbps of bandwidth and the transmission latency is close to 60 ms. This means that a network packet that is transmitted between two different clusters performs three router hops. The packet round trip delay is given by Equation 4.1.

$$RTT_{WAN} \approx 2 \times (2 \times ClusterLatency + 2 \times WANLatency) \tag{4.1}$$

Therefore, accordingly to the configuration described, the transmission time, T, of a network packet between sites on different clusters would be: $T \approx RTT_{WAN}/2 \approx 2 \times (0.12 \times 2 + 60 \times 2)/2 \approx 120ms$. Figure 4.2, depicts this configuration.

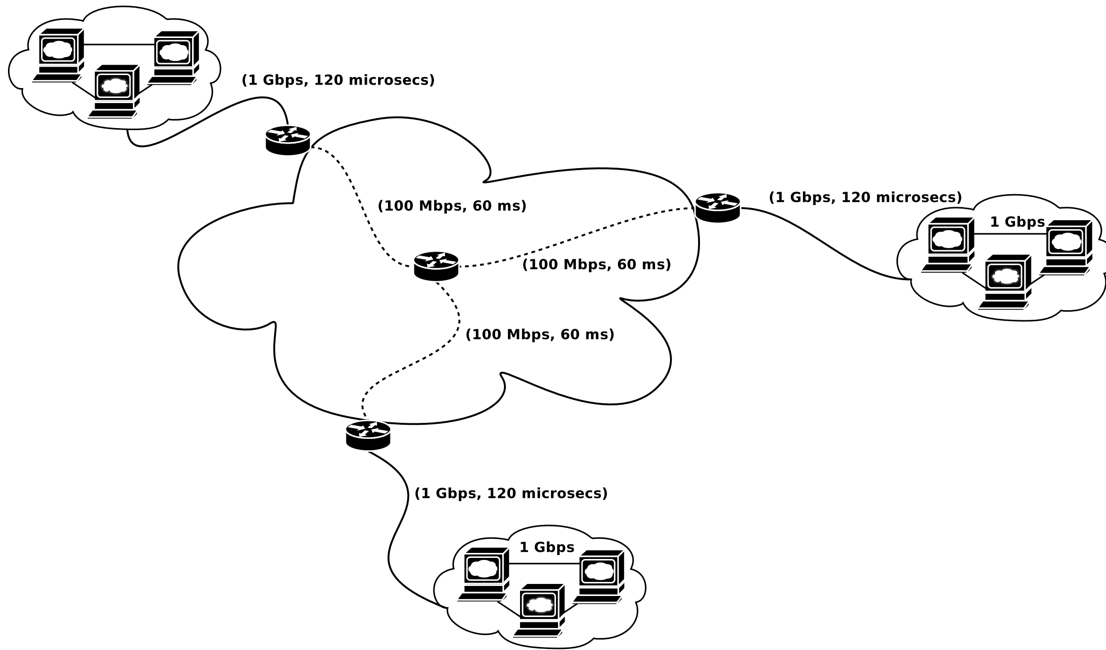


Figure 4.2: Wide area network configuration.

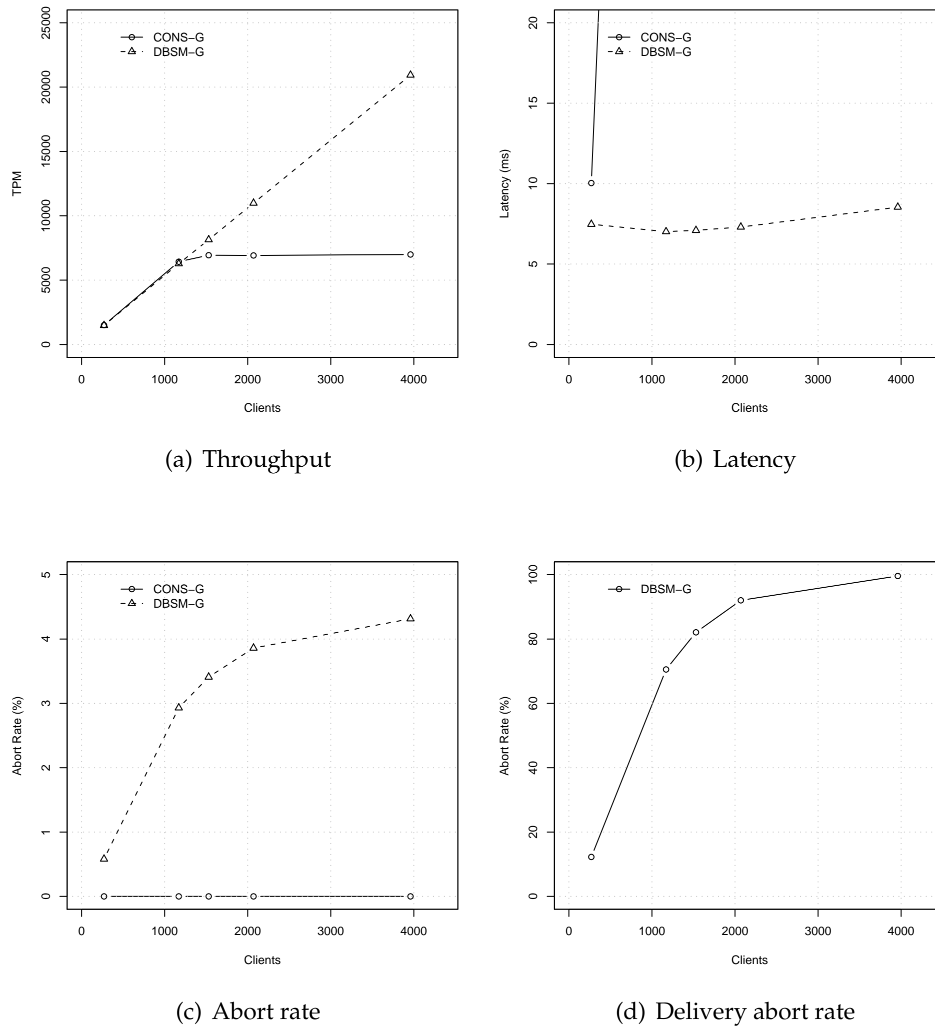
	Serializable			**Snapshot Isolation Level**		
Class./Trans	Neworder	Payment	Delivery	Neworder	Payment	Delivery
Warehouse	x	x			x	
District	x	x		x	x	
Customer	x	x	x		x	x
Item	x					
Stock	x			x		
Orders	x		x	x		x
OrderLine	x		x	x		x
NewOrder	x		x	x		x
History		x			x	

Table 4.1: Definition of coarse conflict classes for each transaction type in TPC-C.

4.3 Results: Optimistic vs Conservative

4.3.1 Coarse Grain

The first study evaluates the conservative and the DBSM approaches without exploiting any application specific details and thus in a configuration that can easily be automated. In the conservative approach, each table is considered to specify a conflict class, which can actually be easily extracted from the SQL code. The resulting conflict classes and conflict relations among transactions types are shown in the "Serializable" column of Table 4.1. Regarding the DBSM, special attention needs to be payed to read-set sizes since the propagation of large read-sets may be impractical. An immediate workaround to this problem is to set a limit for the read-set size over which the whole table is used. In the TPC-C, this results in transactions of type Delivery always being marked as reading



(a) Throughput (b) Latency

(c) Abort rate (d) Delivery abort rate

Figure 4.3: Performance measurements in a LAN with coarse granularity.

the entire OrderLine table. All others access only a small number of items.

Figure 4.3, presents performance measurements in the LAN scenario. It can be observed in Figure 4.3(a), that the DBSM protocol with optimistic execution apparently scales much better to a large number of clients than the conservative protocol. As shown by Figure 4.3(b), the bottleneck in the conservative protocol translates in very large queueing latencies.

This result is highlighted in Figure 4.4 that decomposes latency as seen by a client, and shows that the impossibility to concurrently process transactions that potentially conflict leads to queueing delays which grow very rapidly with the number of clients connected.

However, as seen in Figure 4.3(c), the good throughput of the DBSM is achieved at the expense of a number of aborted transactions. This is especially worrisome since the 4% of transactions being aborted overall are in fact all Delivery transactions as shown in

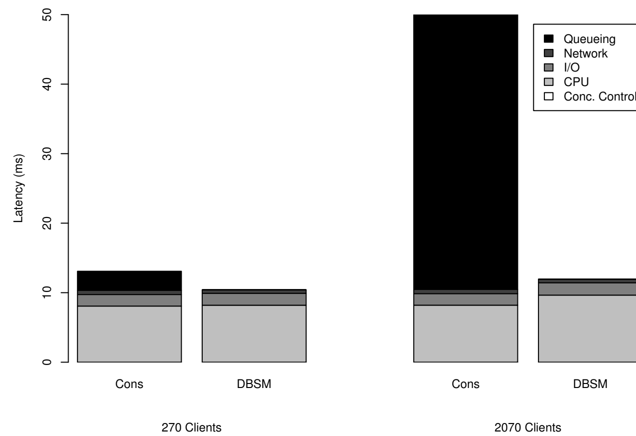


Figure 4.4: Detailed profiling in a LAN with 270 clients.

Figure 4.3(d). Therefore, even if such transactions can be resubmitted, there is a very low probability of ever being executed.

Conclusion: Results show that neither DBSM nor CONS protocols scale, without application specific configuration, to a large number of clients with an OLTP load, even with plenty of resources in a LAN.

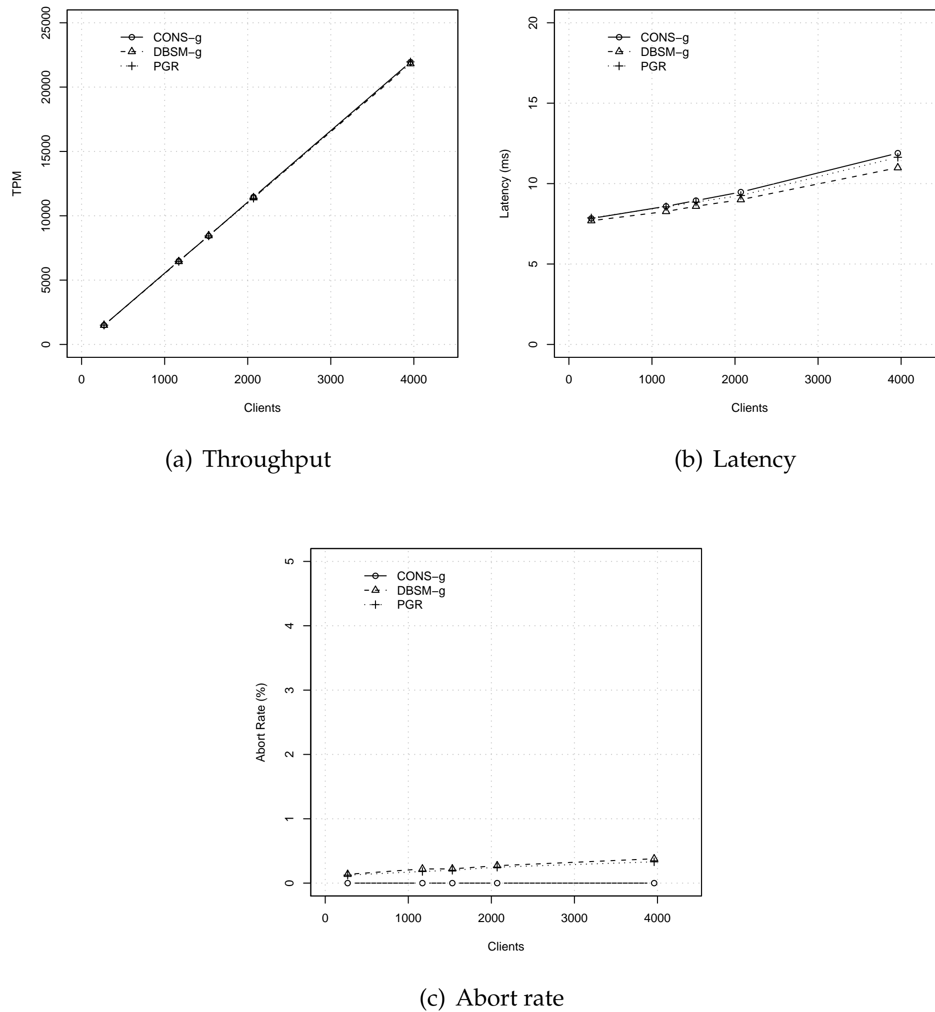
4.3.2 Fine Grain

To reduce the number of conflicts, a finer granularity has to be specified when defining conflict classes for the conservative approach and the read-set extraction in the DBSM. Fine grained conflict classes are obtained by taking advantage of the fact that all tables except Item have references to the Warehouse table and that clients connected to the same node have high locality regarding a specific subset of warehouses.

Although the simulation run considers this granularity for the CONS protocol, it is impractical since one cannot predict beforehand which subset of tuples, a transaction will access, specially for the Neworder and Payment transactions. This renders the approach impractical.

In the optimistic protocol, this granularity is practical, because the read-set is extracted during the transaction execution, hence there is no need to predict it. But if there is no hotspots this approach is worthless.

These optimizations are also compared with the PGR protocol which can use the exact read-set by centralizing certification of each transaction. The results are presented in Figure 4.5. It can be observed that all approaches produce approximate results with minimal differences in latency and abort rate. Network usage is also low, showing that the overhead incurred by the DBSM when sending the read-set is offset by requiring only a



(a) Throughput (b) Latency

(c) Abort rate

Figure 4.5: Performance measurements in a LAN with fine granularity.

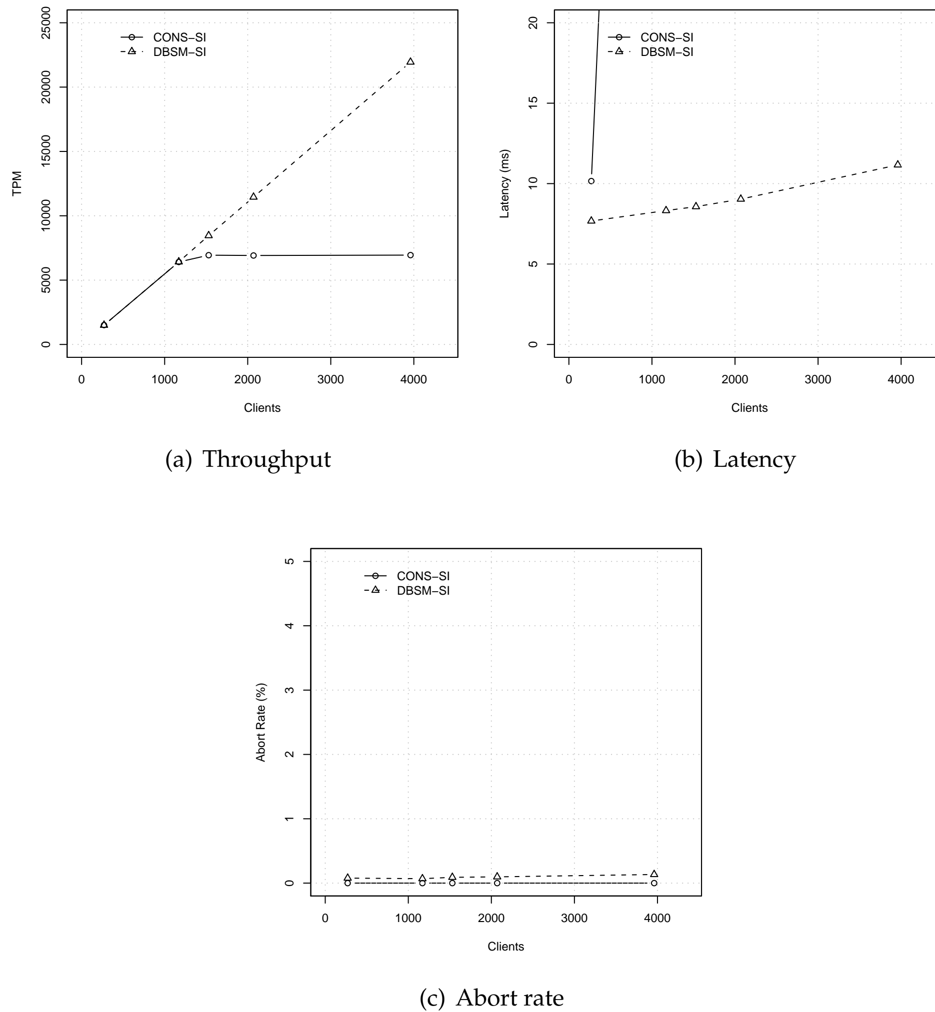
single communication step.

Conclusion: When an appropriate grain can be defined, CONS and DBSM, are equally suited as PGR for an OLTP load in a cluster.

4.3.3 Snapshot Isolation

An alternative approach to avoid synchronization conflicts is to relax the correctness criterion to snapshot isolation [BBG+95] which only considers Write-Write conflicts.

In the DBSM approach, all the concerns previously discussed about the size of the read-set are avoided. As Figure 4.6 shows, it turns out that this alternative has also a benign impact on the performance of the DBSM approach, reducing the number of aborted transactions. Moreover, this is a very appealing alternative, as it avoids all configuration



(a) Throughput (b) Latency

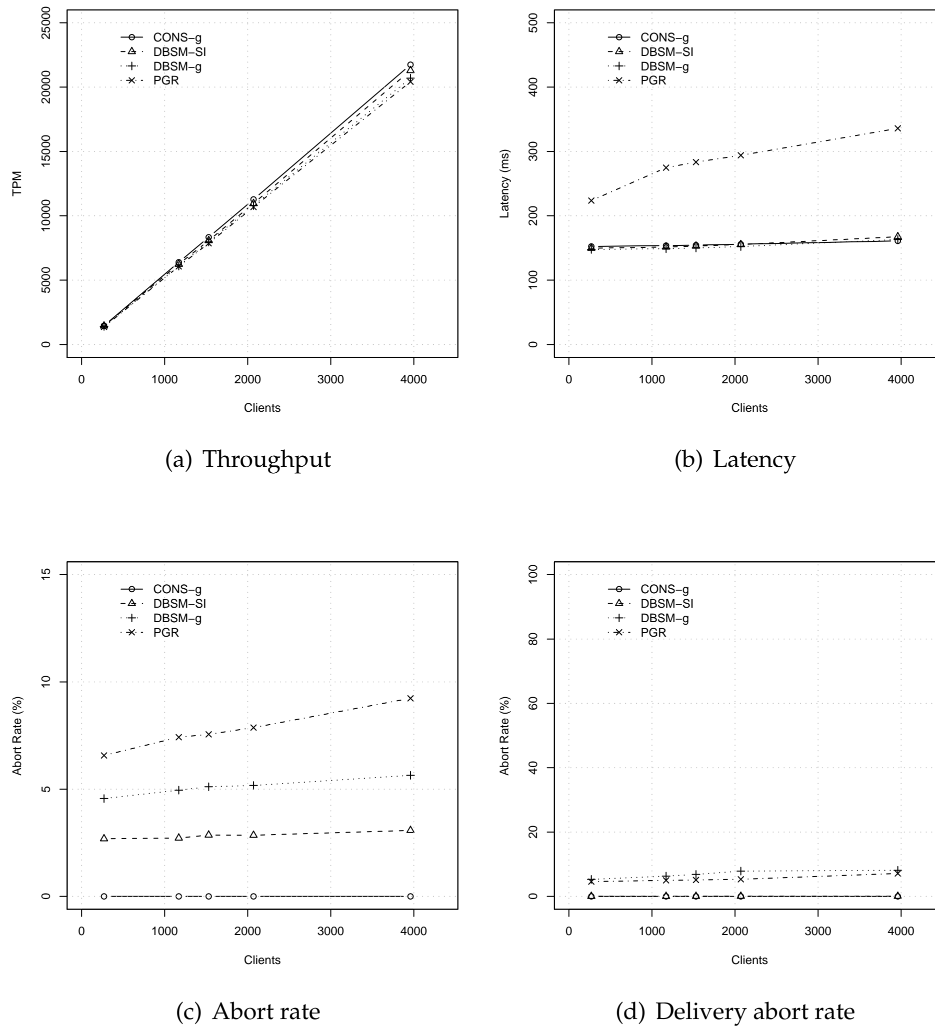
(c) Abort rate

Figure 4.6: Performance measurements in a LAN with snapshot isolation.

issues. Under snapshot isolation the DBSM and PGR protocols become the same.

Unlike the DBSM, the conservative approach does not benefit from the snapshot isolation criterion, exhibiting the same latency as in the coarse grain study. In the "Snapshot Isolation Level" column of Table 4.1 the new conflict relations among the transactions are depicted. Regardless of their type, all update transactions still conflict and thus have to be sequentially executed.

Conclusion: When using snapshot-isolation, DBSM presents a reduced abort rate, while the CONS protocol, despite the relaxed correctness criteria, still suffers from conflict penalties.



(a) Throughput (b) Latency

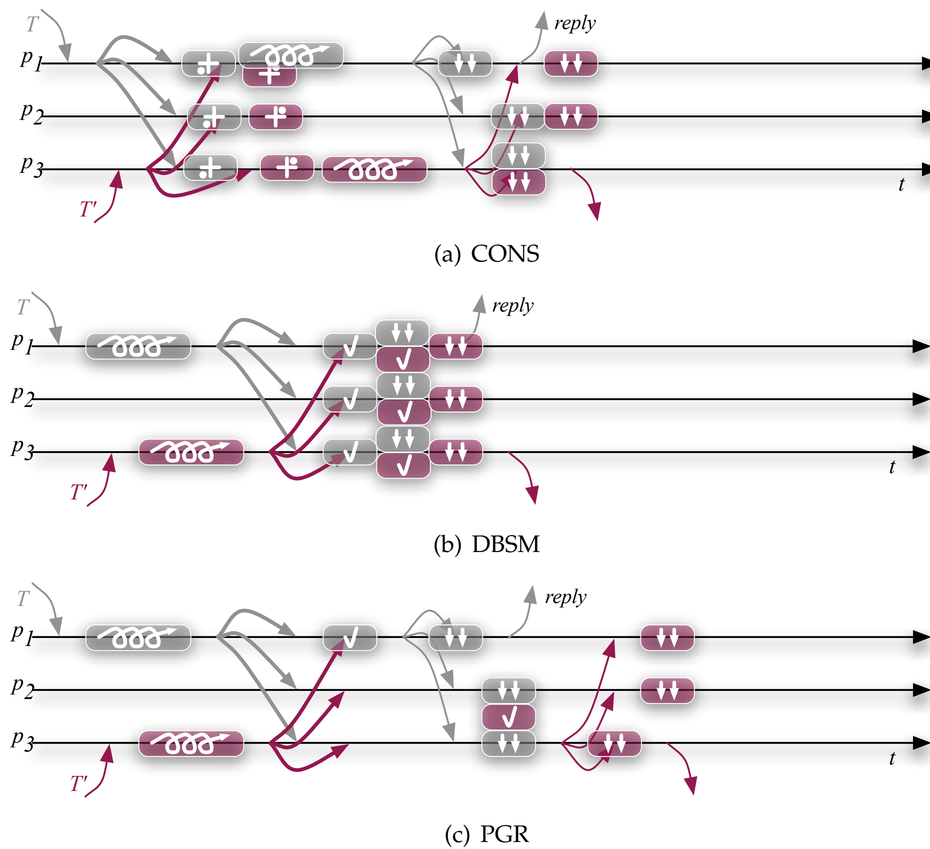
(c) Abort rate (d) Delivery abort rate

Figure 4.7: Performance measurements in a WAN.

4.3.4 Wide Area

It is also interesting to observe how the proposed approaches scale to interconnected clusters in WAN. Wide area networks introduce higher communication latency when compared to LANs. Consequently, transactions take more to conclude, resulting in a higher probability of concurrent transactions in the system, thus higher probability of conflicts.

In a protocol based on distributed locking, the influence of latency can potentially be very large, if a node has to wait that all other nodes enter and leave a critical section plus the time it takes to pass the authorization around. In contrast, when using active replication [Sch93, GS97], the only overhead is encapsulated in the total order multicast protocol and no additional synchronization is required. Ideally, a database replication protocol based on total order multicast would be able to achieve the same goal.



(a) CONS

(b) DBSM

(c) PGR

Figure 4.8: Handling concurrent transactions.

To study the impact of the communication latency overhead when considering WANs, the best performers in the previous scenarios were chosen. Their performance is presented Figure 4.7. Although Figure 4.7(a) shows that throughput scales equally well, Figure 4.7(b) shows that the additional communication step, incurred by PGR, when centralizing certification results in a large increase in latency. This has also an impact in the overall abort rate in Figure 4.7(c), which is higher than with other optimistic approaches. Nevertheless, counterpointing results from Figure 4.3, Figure 4.7(d) shows that no single transaction type exhibits high abort rates, hence, if one chooses to resubmit the aborted transactions there is a high probability of a successful execution.

Figure 4.8, depicts the conservative and optimistic protocols handling the execution of two concurrent non-conflicting transactions. In the CONS protocol (Figure 4.8(a)), once the transactions are ordered, all steps of the protocol are executed concurrently therefore corresponding to the desired behavior.

Regarding the optimistic approaches, one can see that in the DBSM (Figure 4.8(b)) the transactions' execution can always be carried in parallel while the certification procedure needs to be done sequentially. Once the certification is finished, since the transactions do not conflict, the updates may be incorporated concurrently. The DBSM therefore incurs in the certification procedure overhead. However, the certification execution time is usually

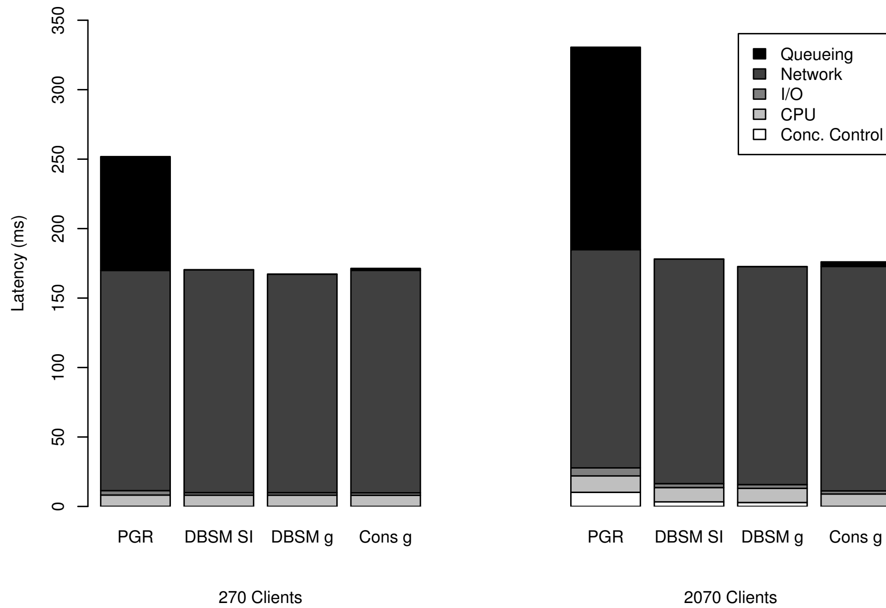


Figure 4.9: Detailed profiling of the Neworder transaction in a WAN.

negligible.

In contrast, the PGR protocol (Figure 4.8(c)) is penalized by the supplemental reliable multicast. Although the transactions' execution can be done in parallel too, the certification of T' (ordered after T) can only be done once p_3 knows the outcome of T. That is, the latency of the reliable multicast of T is incorporated in the response time of T'. Figure 4.9, presents the average latency in each execution step of a Neworder transaction. Queueing stands for the time a transaction is put on-hold in the replication protocol, and the other legends are self explanatory. This figure clearly depicts the extra queueing overhead, in the PGR protocol. Queueing side effects can be further worsened as more concurrently executing transactions exist.

Conclusion: Results show that both, DBSM and CONS family of protocols, behave similar in wide area networks. On the other hand, PGR presents additional queueing overhead, leading to increased execution latency and higher abort rates.

4.4 Results: Fault-injection and DBSM

In this section the simulation model is used to evaluate the performance and dependability of the DBSM prototype implementation.

Fault type	Parameters	Implementation
Clock drift	rate	Scheduled events are scaled up (*i.e.* postponed) and elapsed durations measured are scaled down by the specified rate.
Scheduling latency	distribution	A randomly generated delay is added to events scheduled in the future (*i.e.* in which the process is suspended and scheduled back).
Random loss	rate	Each message is discarded upon reception with the specified probability. Models transmission errors.
Bursty loss	average burst lengths	Alternate periods with randomly generated durations in which messages are received or discarded. Models congestion in the network.
Crash	time	A node is stopped at the specified time, thus completely stopping interaction with other nodes.

Table 4.2: Types of faults injected.

4.4.1 Correctness and Performance

The performance of the termination protocol (set of DBSM and group communication instructions) is assessed by conducting simulations in local area with three sites only. Faults are injected and the termination performance is measured.

Fault injection creates adverse conditions causing message losses or high jitter on message processing time. Table 4.2, exhibits a detailed overview of the type of faults injected into the system. Faults are injected by intercepting calls in and out of the runtime as well as by manipulating model state.

The evaluation of the results is two-fold. First, to ensure that all operational sites must commit exactly the same sequence of transactions by comparing logs off-line after the simulation has finished. This condition has been met in face of all types of faults listed in Table 4.2 and ensures the safety of the approach and of the prototype implementation in maintaining consistency. Second, to measure the impact of faults on the performance of the termination protocol and its ultimate consequences at the transaction level. Besides crashes, which as expected have a profound impact in performance by disconnecting a number of clients, the types of faults in Table 4.2 causing more performance degradation are those causing message losses. Figure 4.10, plots the empirical cumulative distribution functions (ECDF) of certification latency with 1000 clients. Note the logarithmic scale in the x-axis. It can be observed that random loss of 5% of messages has much more impact than the same amount of loss in bursts of average length of 5 messages (uniformly distributed). The long tail of the distribution indicates that a small number of transactions

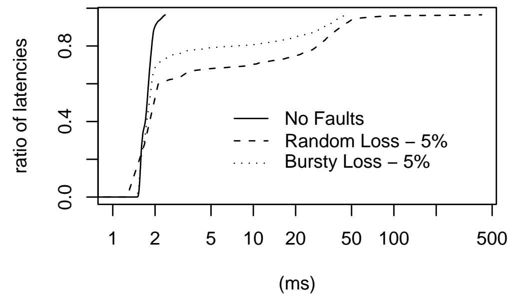


Figure 4.10: Certification latency (fault injection).

Run	Usage
No Faults	1.22
Random Loss	1.90
Bursty Loss	1.89

Table 4.3: Protocol CPU usage (%).

is taking as much as 10 times more, in the termination protocol, than before. Table 4.3 shows also an increase in CPU usage by real jobs, showing the extra work by the protocol in retransmitting messages.

The impact of faults in the quality of service provided to the application should also be measured by the number of aborted transactions presented in Table 4.4. This is explained by the extra time spent in the termination protocol. In fact, the random loss tail corresponds to the group protocol blocking a few times for short periods during the simulation run. Blocking is caused by a combination of three factors:

- The group protocol enforces fairness by ensuring that each process can only own a share of total available buffering.
- Using a fixed sequencer for ordering messages. This leads to a much larger number of messages being multicast by one of the participant processes.
- Each round of the stability detection mechanism can only garbage collect contiguous sequences of messages received by all participants. As loss is injected independently at each participant, the common prefix of messages received by all processes is dramatically reduced, even with loss rate as low as 5%. This slows down garbage collection.

It is therefore likely that the buffer share of the sequencer process is exhausted and the whole system blocked temporarily waiting for garbage collection. The problem is

	3 Sites/1000 Clients		
Transaction	No Faults	Random Loss - 5%	Bursty Loss - 5%
Delivery	1.41	9.84	4.46
Neworder	1.46	3.38	1.63
Payment	12.78	22.54	14.15
Orderstatus	2.43	2.93	1.82
Stocklevel	0.00	0.00	0.00
All	6.72	11.94	7.96

Table 4.4: Abort rates with 3 sites and 1000 clients (%).

mitigated by increasing available buffer space or by allocating a dedicated sequencer process. The ideal solution would be to avoid the centralized sequencer.

Note also in Figure 4.10 that the loss of 5% of messages results in delaying 30% to 40% of messages at the application level. This is a consequence of the total order required by DBSM. This result suggests that relaxing the requirement for total order [PS99] is necessary for efficient deployment in wide area networks.

Conclusion: The certification performance is not affected in any way due to high jitter on processing messages. On the other hand when there are message losses the certification latency increases 10 times for 20% of the messages. Nonetheless, the degradation of the system as a whole is not directly proportional to the latency increase, as it only shows a slight increase in the abort rate.

4.4.2 Performability

Section 4.4.1 presented evaluation of the termination protocol performance when faults were injected. This section extends the previous one, by assessing how well do the DBSM-g (fine granularity) and DBSM-SI (snapshot isolation) protocols variations degrade under faulty conditions. This is done by conducting a *performability* study.

Performability embraces two concepts: performance and dependability. It is a composite measure which is often used to assess if a system is able to degrade gracefully in the presence of faults. If a system degrades gracefully, it tolerates faults, appearing to be working normally. The study presented in this section relates to the fault injection by dropping messages, either randomly or where there are bursts.

The most representative transaction in the TPC-C benchmark is the one that stands for the arrival of a new order to the system. Then, Neworder transactions are a good indicator on how the system is performing. Therefore, one is interested in measuring the impact on these transactions when the system is facing adverse conditions. In other words, how graceful does the execution of Neworder transactions degrade, in the presence of faults, namely when there are packet losses? Having to retransmit the missing

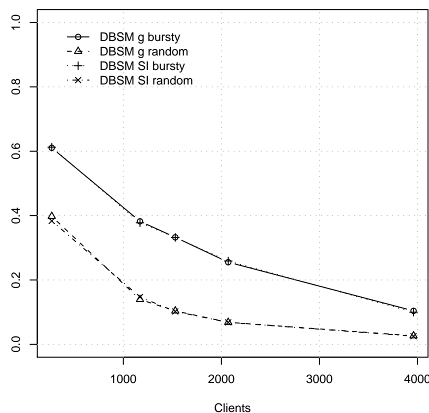
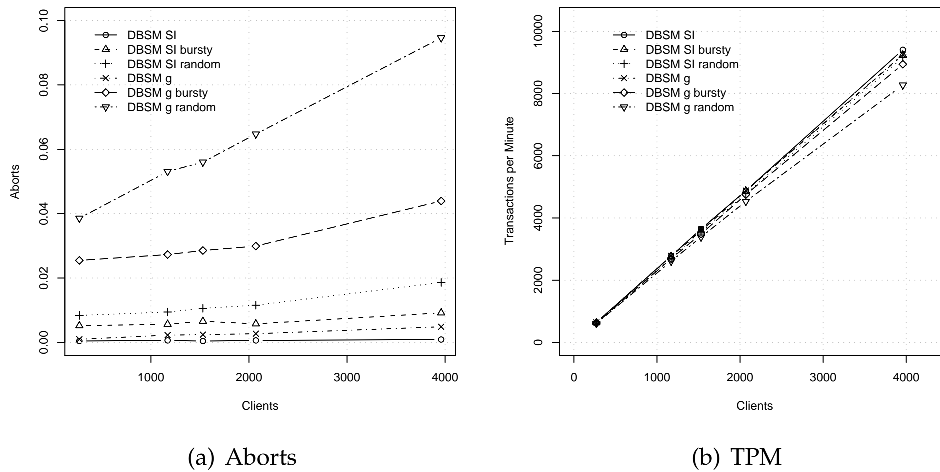


Figure 4.11: Probability of committing a Neworder transaction within the period of time τ.

packets, implies that the replication latency increases, which directly changes the average response times for the incoming new orders. The performability metric considered to conduct the study is the system's ability to commit a Neworder within the period of time needed to commit 90%, of the same kind of transactions, when there are no faults at all [TM96]. Put differently, given L, the average execution latency of a Neworder transaction, and τ, as the percentile 90 of the commit time in the faultless system, the metric considered is the probability: $P(L \leq \tau)$.

DBSM fine granularity and DBSM with snapshot isolation consistency criteria, both in the LAN environment described in Section 4.2, are the scenarios in which *random* and *bursty* drops of messages are injected. Figure 4.11, depicts the system behavior from the referenced performability metric point of view. As expected, as the load increases, the probability of committing a Neworder within the τ period diminishes. As the system suffers more from retransmissions, the probability decreases almost exponentially as load is increased. The results show that random drops are worse tolerated than bursty drops. Note that when there is a light load (270 clients) it is almost 20% worse. Under heavy load, the system performability values tend to be the same, almost zero, both in the presence of bursty and random drops.

Although, from the user point of view, an increase in latency, in a milliseconds scale, may not be perceptible, such growth may influence the response of the entire system, directly in terms of abort rate and consequently in terms of throughput. In the case that there are bursty drops, the probability of a Neworder transaction commits within the τ period decreases from 60% to 10%. In this case, the abort rate for the two different variations of the DBSM protocol remain below the 5%. On the other hand, if one considers random drops, the probability decreases from 40% to almost 5%. Therefore, the abort rate will increase due to the higher probability of the execution exceed given τ. However, note that none of the protocol variations shows an abort rate above 10%, even



(a) Aborts (b) TPM

Figure 4.12: Performance comparison.

under heavy load (3960 clients). Figure 4.12(a), depicts the abort rate.

Figure 4.12(b), depicts the throughput of successful new orders per minute that the system is able to achieve. It shows that there is only one situation in which performance degradation is perceptible. This is the case of *DBSM g random*. All the others are not affected considerably in the presence of faults. In fact, in the cases that snapshot isolation is considered the system does not exhibit degradation at all since, the abort rate remains below 2% and the throughput matches the one in which there are no faults. One may conclude that snapshot isolation is more tolerant to latency variations due to network retransmissions than the other variant (fine grained). The reason is that the SI consistency criteria considers only Write-Write conflicts, therefore, despite the increase of network latency, which leads to a larger period of time in which concurrent transactions may occur, few Write-Write conflicts do really happen. In the fine grained consistency criteria, Read-Write conflicts are detected, hence there is a higher probability that a conflict occur, when increasing the network latency.

A final remark to the performability of the random drop scenarios. When looking at the abort rate for the *DBSM SI* and *DBSM g*, one may find that there is a difference of almost 7%. This difference should be translated into a smaller value in the probability of committing a Neworder transaction when comparing both. What is preventing this from happening is that aborted transactions take less to execute, and release their resources sooner. This way, *DBSM g* system exhibits less contention when compared to the *DBSM SI*. Less contention means less time that transactions have to wait for resources to become available, hence exhibit smaller execution times. As a consequence, the probability of committing a transaction within the τ period of time becomes the same for the two protocols.

Conclusion: It was shown that for the snapshot isolation variant, the system degrades

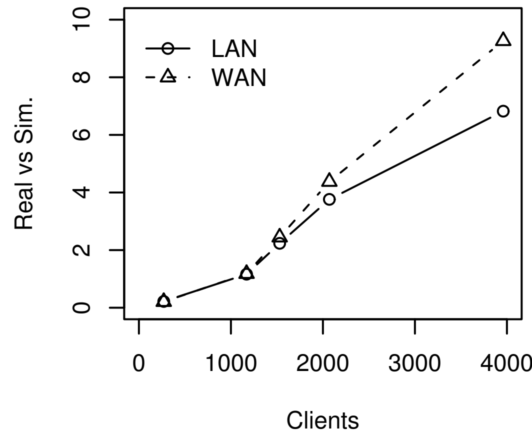


Figure 4.13: Ratio of time required to run the simulation vs. simulated time interval (9 servers/increasing number of clients).

gracefully, both in the presence of bursty and random drops, despite that in the latter case it suffers most. In the fine grained variant, the performance degradation of the system is not perceptible in the presence of bursty drops. However, when random drops are injected, the system is not able to keep up with the performance of the faultless scenarios.

4.5 Simulation Performance

The usefulness of the resulting setup for experimental evaluation of performability, early and often as implementations of replication and group communication protocols evolve, is tightly related to the ratio of time required to run the simulation in a single host to the simulated time interval. Figure 4.13, shows that a cluster of 9 servers and sufficient client resources to open 4000 client sessions generating up to 12000 tpm (*i.e.*, an additional cluster) can be simulated in less than 8 times the simulated time interval. When simulating 3 clusters of 3 servers interconnected by a WAN, which would be substantially harder and more costly to setup in reality, only a small additional delay is incurred.

Chapter 5

Conclusion

This work addresses the problem of evaluating group-based database replication protocols in a way that is representative and comparable. Previous work was based either on simplistic simulations or toy applications, or was tightly coupled with a single database management system or replication protocol.

However, preparing a fully featured testing environment using real resources, is costly to setup and run and often depend on the availability of the complete target system, on realistic workloads and fault-loads. This is especially difficult when the application targets large clusters or grid systems, and precludes incremental development and early testing of individual components. In fact, testing distributed applications, often requires deploying more than once the target system, including client-side software and hardware. This makes the approach very expensive and limits the possibility to observe the global system state.

The proposed approach builds on the concept of centralized simulation [AC97] as way to combine real implementations of components under study with a simulated environment. The work presented in Section 3.3 does, however, expand it in two different ways:

- The implementation is achieved with minor extensions to an existing standard interface for event driven simulation, thus making it possible to reuse existing models, namely, SSFNet [CLL+99].
- A library based on Java reflection provides a fully transparent interface between implementation components and the simulated environment with minimal programming effort.

Also in contrast with previous work on centralized simulation, the simulated application is not a simple component. Instead, a detailed simulation of a database management

system is used, thus providing to the replication protocols a realistic scenario that encompasses resource usage and synchronization issues. This was achieved in two steps:

- The SSFDb model library created provides abstractions which accurately reproduce the behavior of a real DBMS system. It is comprised of several models which mimic central processing units, storage, database engine scheduler, lock manager and replication manager. It is described in Section 3.2.
- Model tuning is accomplished by calibrating each model component in such a way that the simulation is actually comparable to the real system. A description of the calibration process and a validation of the result is included in Sections 3.4 and 3.5.

The usefulness of the approach is shown in Chapter 4 by presenting and discussing how results obtained with the framework answer to a number of questions. In detail, several replication protocols (DBSM, Postgres-R and Conservative) are compared in different communication scenarios.

Results show that in local area networks, if a simple configuration is used, the optimistic family of protocols would always ending up aborting a specific kind of transaction. On the other hand, the conservative family of protocols would generate very high conflict rates, meaning that queueing effects become dramatic. Therefore transactions would execute in a sequential manner, rendering the system useless due to the latency overhead. Previous results were not able to disclose this.

A second interesting result was obtained when the consistency criteria was set to snapshot isolation. The optimistic family would get good results, while the conservative would still exhibit deficient behavior. The protocols that performed better in local area were also tested in wide area environments, exhibiting identical performance among them. Previous results had not addressed group based replication protocols in WANs.

Finally, a performability study assessed system degradation when faults were injected in two DBSM protocol variants: fine granularity and snapshot isolation. Results show that DBSM fine granularity suffers most when facing packet losses because latency increases, due to packet retransmission, allowing more concurrent transactions to execute. Such phenomena leads to a higher probability of Read-Write conflicts and consequently to higher abort rates.

Despite a small delay introduced when comparing the same testing scenario in simulation against the real system, one notices that simulation is more attractive because it does not require extra hardware/software deployment as it is in the real system.

The incremental development environment provided by the simulation framework is also a great asset. It allows development to start from a collection of pure simulation models that are replaced as real implementations become available. From the initial simulation model, tests may be derived and used to assess if the real implementations are

in conformity to the simulated models that they have replaced. This is quite similar to the unit testing approach, which is however hard to apply to middleware, where initial modeling focus is on performability of the system and not on independent correctness of each component.

5.1 Future Work

Further development is envisioned upon all contributions of this work, namely, on the simulation kernel, on environment models, and on protocol evaluation.

Regarding the MinhaSSF simulation kernel, it is a priority to support realistic simulation of multiple threads running on the same CPU. Besides minor changes to kernel to properly account real time and scheduling, this will require providing simulated models of concurrency control primitives. The resulting package should then be useful to support incremental development of a wide spectrum of middleware for distributed systems. This is being driven by the development of the Appia [MPR01] group communication system also within the GORDA project [Con04].

Also regarding simulation models, a better support for fault-injection within SSFNet is sought. In this sense, one would be able to easily setup faulty processes or faulty network connections. This is required to apply MinhaSSF to the study of other distributed applications, in particular, to large scale event dissemination and peer-to-peer systems. This is being driven by the development of the NeEM protocol in the P-SON [ML05] project.

Finally, the experimental framework described here is being used daily within the GORDA project to evaluate new protocols, new protocol configurations and existing protocols under different environment conditions. In fact, all protocol development at the U. Minho is first done in the context of the MinhaSSF framework and thus new results are added to those in Chapter 4 every day.

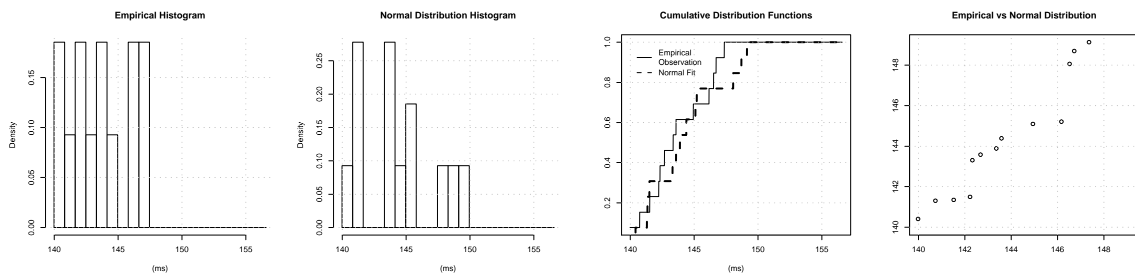
Appendix A

Distribution Parameter Estimation

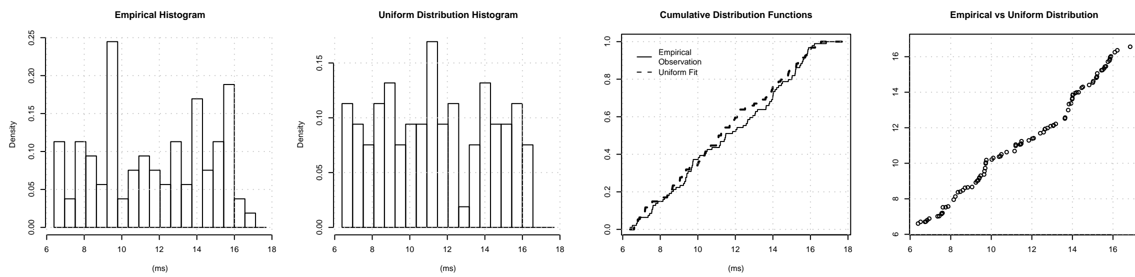
Choosing a generic distribution is accomplished by intuitively fitting a generic distribution to the empirical observation. By looking at the histogram, drawn from the samples, one is able to pick one distribution that the population is following. Having chosen the distribution, the parameters need to be estimated, hence, the *fitdistr* function from *R* [1] is used. This function analyzes the samples and estimates the parameters of the given distribution. In order to assess the goodness of the fit, a Q-Q plot is drawn. In the Q-Q plots presented in this section, values of the empirical observation are plotted against values obtain by sampling the fitted distribution. If the plot shows a straight line, 45 degrees steep, then data obtained from the empirical observation comes from a population that follows the given distribution.

The following figures show the fitting of the CPU and storage times, obtained for each one of the transactions in the TPC-C benchmark.

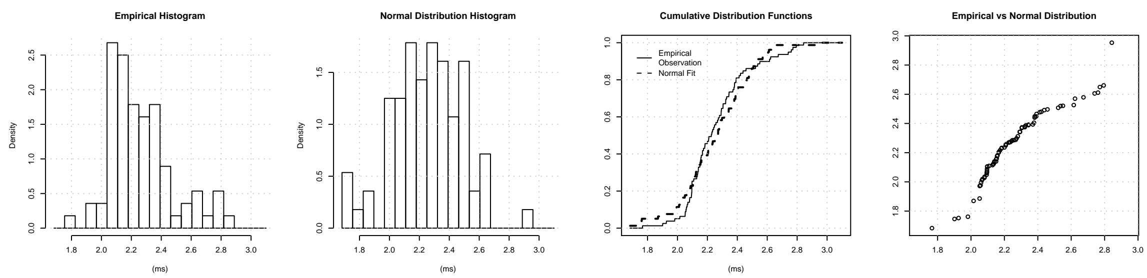
[1] software project for statistical computation (http://www.r-project.org/)



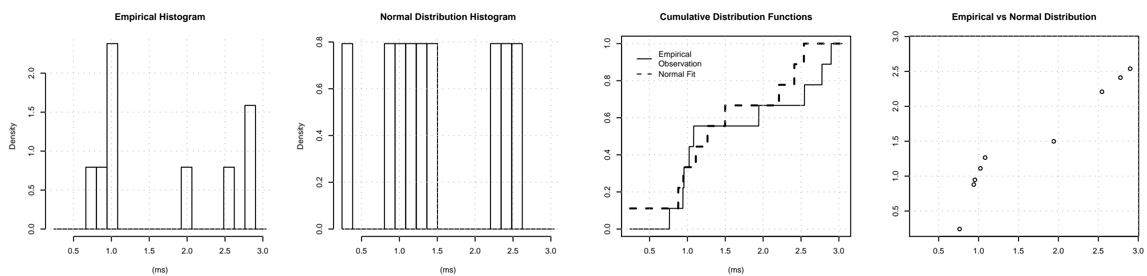
(a) Delivery.



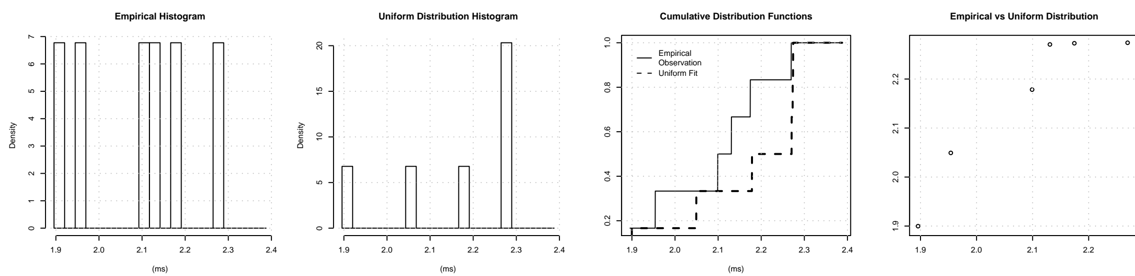
(b) Neworder.



(c) Payment.



(d) Orderstatus.



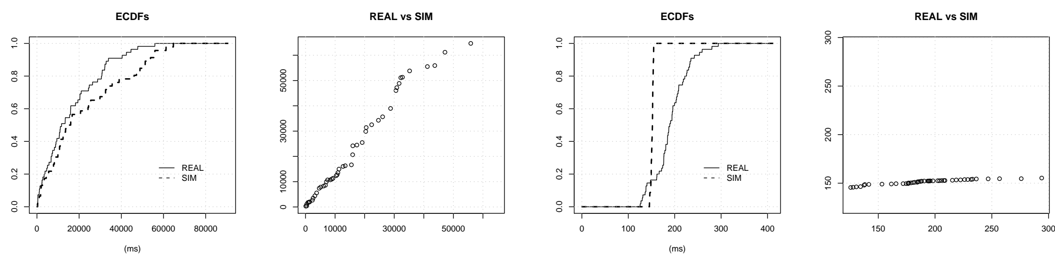
(e) Stocklevel.

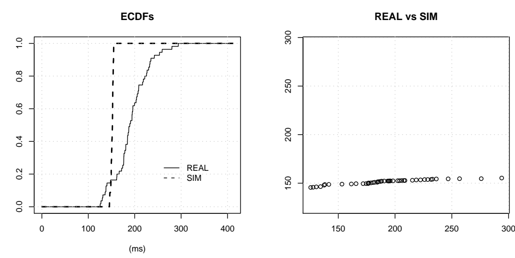
Figure A.1: Processing time fitting.

Appendix B

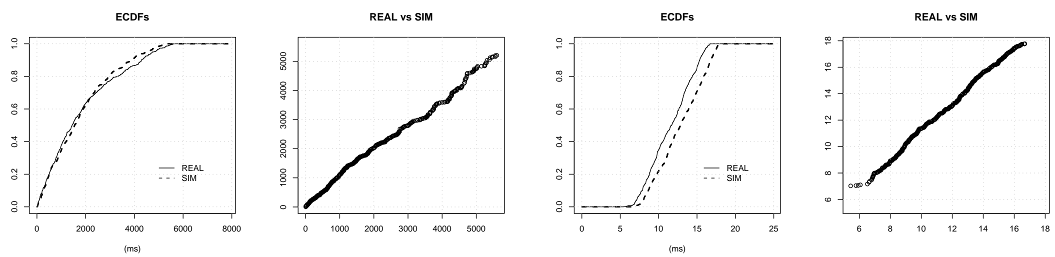
Real vs Simulation Model

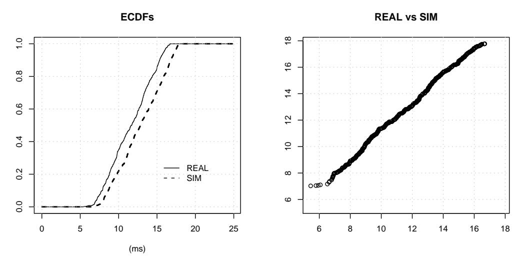
This section details the comparison of the real and simulated run. It compares each transaction in terms of the considered performance metrics: inter-arrival, latency. These are depicted in the following figures.



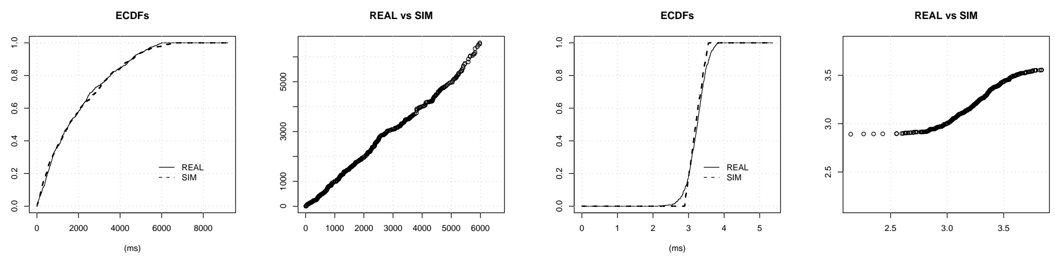


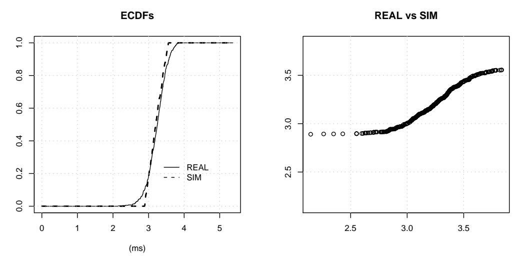
(a) Delivery Inter-arrival. (b) Delivery Latency.



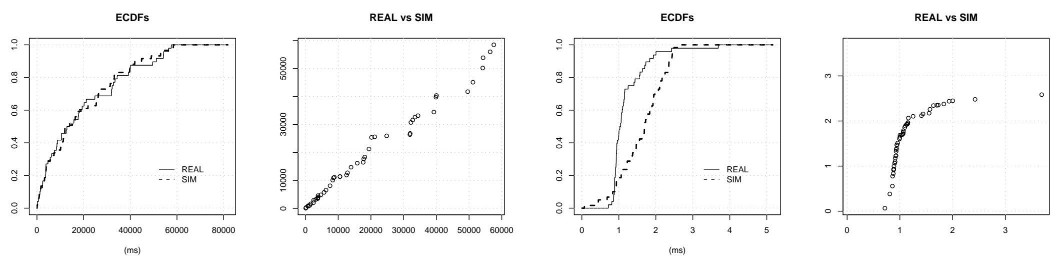


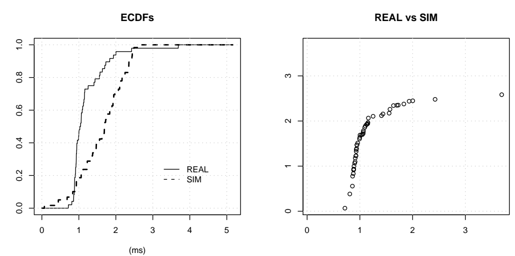
(c) Neworder Inter-arrival. (d) Neworder Latency.



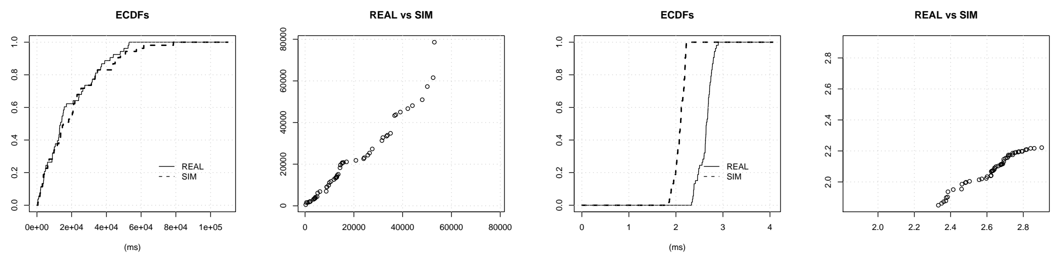


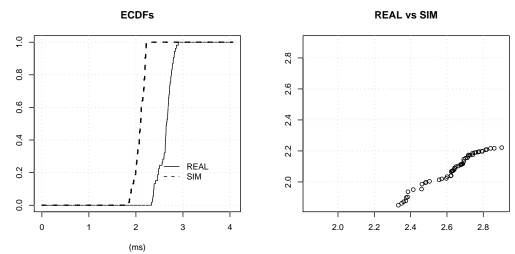
(e) Payment Inter-arrival. (f) Payment Latency.





(g) Orderstatus Inter-arrival. (h) Orderstatus Latency.





(i) Stocklevel Inter-arrival. (j) Stocklevel Latency.

Figure B.1: Transaction performance comparison.

Bibliography

[AC97] G. Alvarez and F. Cristian. Applying simulation to the design and performance evaluation of fault-tolerant systems. In *16th Symposium on Reliable Distributed Systems (SRDS'97)*, 1997.

[ACL85] R. Agrawal, M. Carey, and M. Livny. Models for Studying Concurrency Control Performance: Alternatives and Implications. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data*, 1985.

[ACL87] R. Agrawal, M. Carey, and M. Livny. Concurrency control performance modeling: Alternatives and implications. *ACM Transactions on Database Systems*, 1987.

[ACM06] ACME. Automated configuration management environment, 2006.

[Alm03] W. Almesberger. umlsim - a uml-based simulator. In *Linux Conference Australia*, 2003.

[AT02] Y. Amir and C. Tutu. From total order to database replication. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[BBG+95] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data (SIGMOD '95)*, 1995.

[BHG87] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[BINN00] J. Banks, J.S. Carson II, B. L. Nelson, and M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, 2000.

[BJ87] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, 1987.

[BMST93] Navin Budhiraja, Keith Marzullo, Fred B. Schneider, and Sam Toueg. The primary-backup approach. 1993.

[BS01] K. Buchacker and V. Sieh. Framework for testing the fault-tolerance of systems including os and network aspects. In *In Proc. High-Assurance System Engineering Symp. (HASE'01)*, 2001.

[BvR94] K. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1994.

[CB02] T. Connolly and C. Begg. *Database Systems: A Practical Approach to Design, Implementation and Management 3rd Ed.* Addison-Wesley Longman Publishing Co., Inc., 2002.

[CLL+99] J. Cowie, H. Liu, J. Liu, D. Nicol, and Andy Ogielski. Towards realistic million-node internet simulation. In *Intl. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[Con04] GORDA Consortium. Gorda - open replication of databases. http://gorda.di.uminho.pt/consortium, October 2004.

[Cor01] Winter Corporation. Scalable network storage: Convergence of san and nas with highroad. http://www.jos.com.my/events/doc/(EMC)HighRoadWinterPaper.pdf, 2001.

[Cou01] Transaction Processing Performance Council. TPC Benchmark™ C standard specification revision 5.0, February 2001.

[Cow99] J. Cowie. *Scalable Simulation Framework API Reference Manual*, 1999.

[Dal02] P. Dalgaard. *Introductory Statistics with R*. Statistics and Computing. Springer, 2002.

[DML] DML specification. http://ssfnet.org/SSFdocs/dmlReference.html.

[DSU04] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 2004.

[ea97] D. Bates et al. The r project for statistical computing. http://www.r-project.org/, 1997.

[EMU] Emulab. http://www.emulab.net/.

[Fac06] Facilita. Facilita forecast, 2006.

[gcs01] Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 2001.

[GS96] R. Guerraoui and A. Schiper. Fault-tolerance by replication in distributed systems. In *Proceedings of the 1996 Ada-Europe International Conference on Reliable Software Technologies*, 1996.

[GS97] R. Guerraoui and A. Schiper. Software-based replication for fault tolerance. *IEEE Computer*, (4), 1997.

[Guo98] K. Guo. *Scalable Message Stability Detection Protocols*. PhD thesis, 1998.

[iSS] iSSF homepage. 2003.
http://www.crhc.uiuc.edu/~jasonliu/projects/issf/.

[KA00] B. Kemme and G. Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, 2000.

[KT91] M. Kaashoek and A. Tanenbaum. Group communication in the Amoeba distributed operating system. In *Proc. the 11th International Conference on Distributed Computing Systems ICDCS*, pages 222–230, Washington, D.C., USA, May 1991. IEEE CS Press.

[LNPP99] J. Liu, D. Nicol, B. J. Premore, and A. L. Poplawski. Performance prediction of a parallel simulator. In *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*, 1999.

[lSPEC05] SPEC logo Standard Performance Evaluation Corporation. Standard performance evaluation corporation (web 2005). http://www.spec.org/web2005/, 2005.

[ML05] U. Minho and F.C.U. Lisboa. Probabilistically-structured overlay networks. http://pson.lsd.di.uminho.pt/, 2005.

[MPR01] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proceedings of The IEEE 21st International Conference on Distributed Computing Systems*, 2001.

[NL97] D. M. Nicol and X. Liu. The dark side of risk (what your mother never told you about time warp). In *PADS '97: Proceedings of the eleventh workshop on Parallel and distributed simulation*, 1997.

[NL02] D. M. Nicol and J. Liu. Dartmouth ssf, 2002.

[NS2] The network simulator - ns-2, 2006.

[OV99] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall International, Inc., 1999.

[Ped99] F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, 1999.

[Pet04] M. Pettersson. Linux performance counters. http://user.it.uu.se/ mikpe/linux/perfctr/, 2004.

[PGS98] F. Pedone, R. Guerraoui, and A. Schiper. Exploiting atomic broadcast in replicated databases. In *Proceedings of EuroPar (EuroPar'98)*, 1998.

[PGS03] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 2003.

[Pla] Planetlab. http://www.planet-lab.org.

[PMJPKA00] M. Patino-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *Proceedings of the 14th International Conference on Distributed Computing*, 2000.

[PRO03] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable multicast: Definition implementation and performance evaluation. *Special Issue of IEEE Transactions on Computers on Reliable Distributed Systems*, 2003.

[PS99] F. Pedone and A. Schiper. Generic broadcast. 1999.

[PS03] F. Pedone and A. Schiper. Optimistic atomic broadcast: A pragmatic viewpoint. 2003.

[PTK94] S. Pingali, D. Towsley, and J. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, May 1994.

[RBDH97] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod. Using the simos machine simulator to study complex computer systems. *Modeling and Computer Simulation*, 1997.

[Sch93] F. Schneider. Replication management using the state-machine approach. In S. Mullender, editor, *Distributed Systems*, chapter 7. Addison Wesley, 1993.

[SPMO02] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *Proc. 21st IEEE Symposium on Reliable Distributed Systems*, 2002.

[SPS+05] A. Sousa, J. Pereira, L. Soares, A. Correia Jr., L. Rocha, R. Oliveira, and F. Moura. Testing the dependability and performance of GCS-based database replication protocols. 2005.

[SS93] A. Schiper and A. Sandoz. Uniform reliable multicast in a virtually synchronous environment. May 1993.

[SSF] SSF research network.
http://www.ssfnet.org/homePage.html.

[Tho98] Alexander Thomasian. Concurrency control: methods, performance, and analysis. *ACM Comput. Surv.*, 1998.

[TM96] A.T. Tai and J. F. Meyer. Performability management in distributed database systems: An adaptive concurrency control protocol. In *Fourth IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'96)*, 1996.

[Uni06] Cornell University. Testzilla - a framework for the testing of large scale distributed systems., 2006.

[VOTC96] L. Valadares, Rui Carvalho Oliveira, Isabel Hall Themido, and F. Nunes Correia. *Investigação Operacional*. McGraw-Hill, 1996.

[WK05] S. Wu and B. Kemme. Postgres-r(si): Combining replica control with concurrency control based on snapshot isolation. April 2005.

[WLS+02] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. USENIX Association, 2002.

[WPS+00] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, 2000.