# DATABASE PERFORMANCE IMPROVEMENTS USING TRANSPARENT PROXYING OF RELATIONAL DATABASES

**Tonny Gregersen** [*] **Lars Chr. Hausmann** [*] **Peter Korsgaard** [*]
**Claus Thomsen** [*]

[*] *Department of Medical Informatics and Image Analysis,*
*Aalborg University*

Abstract:
In the process of testing the hypothesis that methods used in distributed systems can be used for improving a centralized database system a proof-of-concept implementation of a Structured Query Language proxy is developed. The developed proxy is optimized for systems which mainly perform search queries. A performance test of the developed system shows that methods used to ensure reliability and availability in distributed systems can be used to increase performance in systems where the number of searches exceed the number of updates, hereby confirming the hypothesis.

Keywords: Database, Performance, Network, Relational Databases, SQL

## 1. INTRODUCTION

This article deals with the analysis, design, implementation, and testing of a Structured Query Language (SQL) proxy used to optimize performance and flexibility of existing centralized database systems. The SQL proxy is developed in order to evaluate the following hypothesis: *Can methods used to ensure high availability, high performance, and high reliability in distributed systems be used to increase performance and flexibility of an existing centralized database system, while not changing any elements of the existing system.*

The motivation for the above hypothesis comes partly from the performance limits imposed by hardware costs on high end centralized systems and partly from the scalability and cost-efficiency of distributed systems (Schroeder, 1994). However as noted by Silberschatz et al. (1997, chapter 16) the benefits of distributed systems comes at the cost of increased development and management cost. Reflecting on the above considerations, it is evident that a system which could benefit from distributed computing without the added costs would be a priority, and is as such the main goal of this project. Throughout the article it should be noted that the project group has limited this task to systems which are inherently search-intensive and which has a large number of concurrent clients, such as search engines for the Internet.

At present time it has not been possible to find a similar product. During this project the authors of this article has kept in touch with the two foremost Open Source projects in the database scene (the PostgreSQL project (PostgreSQL, Org., 2000a) and the MySQL project (MySQL, Inc., 2000a)), but none of these have been able to show a competing solution. It should however be noted that the MySQL project has started implementing features to enable support for a distributed database system (MySQL, Inc., 2000b). Likewise PostgreSQL Inc. is currently working on building a distributed database system on top of the PostgreSQL database server (PostgreSQL, Inc., 2000).

As shown in figure 1 the proposed solution to the hypothesis consists of a SQL proxy designed to mimic the interfaces for both the client and server parts of an existing database system and thereby ensuring transparency. The functionality of the proxy is centered on distributing queries from clients to the currently

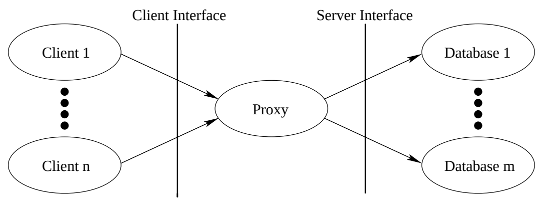

Fig. 1. Basic structure of the proxy solution

least busy database server, all while maintaining consistency of all database servers in the cluster.

In this article it will be shown that the proxy approach is able to provide a performance gain regarding read-only queries compared to an existing centralized database. It will also be noted that this performance gain is achieved at the cost of a performance degradation regarding modifications of the database.

## 2. METHODS

From the presentation of the system to be developed the following basic requirements are given:

- The system should be transparent to the clients of the system.
- The system should be able to handle crashes of database servers in the database cluster.

In the following methods to fulfill the requirements mentioned above will be described.

### 2.1 *Distribution of data*

Silberschatz et al. (1997, chapter 18) suggests that there are two principal methods which can be used to distribute data among a number of sites, namely the techniques of replication and fragmentation. One of the two techniques can either be used exclusively, or a combination of both can be used.

Using replication, data is mirrored on several database servers. This has the advantage of increasing the availability of data, since the system might continue to operate after one or more database servers has failed. An added advantage is also the increased performance of searches as these can be conducted in parallel. On the other hand updates become more extensive, since several database servers have to be updated.

Fragmentation can be divided into two sub-categories: Vertical and horizontal fragmentation.

- In vertical fragmentation the database is divided into several sub-databases, each containing one or more relations. This has the advantage that updates on relations located at a single database does not require the overhead of communication between the database servers of the cluster. Unfortunately, queries which require data from several relations become more extensive.
- Horizontal fragmentation divides the tuples into several subspaces of the database. This way each database server contains a number of tuples from each relation. The advantage of this type of fragmentation is that queries requiring only the subset available on one database server can be conducted without interaction from the rest of the cluster. Furthermore, searches in the database can be split into several sub-searches each running in parallel on the servers of the cluster.

When designing a system in which some or all data are replicated among a number of hosts, a major concern is to ensure consistency of data. In single site systems data consistency can be ensure by using transactions (Weihl, 1994) and in distributed systems by the use of commit protocols (Silberschatz et al., 1997, p. 604-612). Performance optimization of the commit protocol is however not of high priority because of the low number of updates expected in the target system.

### 2.2 *Fault-tolerance*

In a distributed system the individual servers can fail independently. Failures are by Schneider (1994) divided into two types: Byzantine failures and fail-stop failures.

- A system experiencing Byzantine failures continues its normal behavior, but exhibits random misbehavior at arbitrary moments. This is not detectable by other parts of the system.
- A system experiencing a fail-stop failure stops its normal behavior, hereby allowing the failure to be detected.

To ensure that the system is fault-tolerant and is able to continue even if some of the database servers in the cluster fails, two different methods to handle the failure of one or more database servers in the database cluster will be described. The described methods are the state-machine and the primary-backup approach.

The state-machine approach is described in more detail by Schneider (1994). The state-machine approach is characterized by the use of a decentralized control structure in which the individual hosts work together to ensure the overall consistency of the cluster. This is done by formalizing the state and commands which changes the state of each unique set of data items within the cluster in what is known as a state machine. The state-machines and their corresponding dataset are replicated among the hosts of the cluster with the number of replicas determined by the desired level and type of fault-tolerance.

Fault-tolerance and consistency among the replicas of each dataset is ensured by the corresponding state-machines upholding the following demands:

**Agreement.** Every non-faulty state-machine replica receives every request.

**Order.** Every non-faulty state-machine replica processes the requests in the same relative order.

The agreement demand can be satisfied by using an agreement protocol among replicas of the same state-machine, which ensures that:

- All non-faulty replicas agree on the same value.
- A replica is only non-faulty, if all non-faulty replicas use its value as the one on which they agree.

The demand of order can be satisfied by using an unique identifier for each request, such as a time-stamp, and having the replicas process requests according to the order of the unique identifiers.

The primary backup method is described more in-depth by Navin Budhiraja and Toueg (1994). The basic idea in primary backup is that the client only communicates with one server, designated the primary. The rest of the servers are designated as backups and they are updated by the primary. If the primary fails a failover occurs in which one of the backups takes over the job of the primary.

The primary-backup method involves less redundant processing and is less costly compared to the state-machine method. The cost for this is that when a failure occurs a request can be lost and additional protocols may be needed to solve this problem.

The two methods described above can be use to ensure the reliability criteria mentioned in the hypothesis. Both methods ensures fault tolerance in that the system can keep running even though some of the database server in the database cluster fails.

## 2.3 *Recovery*

In case a server in the cluster becomes available after experiencing a fail-stop failure, it has to be synchronized with the cluster before it again can become part of the cluster. This can be done using one of several recovery techniques. Among these are the full stop technique, the temporary client technique, and the log based technique. The full stop technique is taken from (Silberschatz et al., 1997, chapter 15), whereas the temporary client and the log based techniques are developed from numerous existing techniques specifically for the task at hand.

In the full stop recovery technique all database servers are put on hold while the crashed database server is updated by making a full copy of the database from one of the database servers which are currently on hold. The technique is simple, but has the unfortunate property that none of the database servers can be used while recovering the crashed server.

In the temporary client technique the crashed database server acts like a client to the system. The database is recovered by making requests to the active database servers. In order to keep consistency all the updates to the active database servers have to be performed on the recovering server as well. This technique has the advantage that it allows access to the active database servers while recovering the crashed server, but this advantage comes at the cost of higher complexity.

In the log based recovery technique a log is kept of all the updates made to the database. The log contains information about which database servers performed the update and a description of the actual update. Using the log a crashed database server can be recovered to a consistent state by searching back through the log, finding the first entry where the server exist in the list of database servers, and going forward through the log performing all the updates stated in the log. The log based recovery technique has the advantage that it minimizes the restoration time by only performing the necessary updates to the crashed server and that it does not need to perform queries to the active database servers opposed to the temporary client technique.

The three methods mentioned above can all be used to restore a database server in the database cluster after a failure.

## 2.4 *System Design*

A list of requirements were given in the introduction and methods needed to uphold these requirements were presented in the beginning of this section. In the following the choice of methods will be presented as a list of additional demands to the developed system.

- In order to maximize the performance gain of search queries the system should use full replication of data between all servers in the cluster.
- The system should ensure consistency of data when modifying the database.
- The system should be able to detect fail-stop failures of the servers in the cluster, but not Byzantine failures as these are considered the responsibility of the individual database servers.
- The system should not be able to restore databases subjected to fail-stop failure.

In order to uphold all demands presented the following system is designed and it will in this context be presented by its component and process architecture.

*Component architecture* The system consist of driver modules for the three external interfaces (one in the status module and two in the database module), a scheduler module which distributes queries from clients between servers and maintains connections to the database servers, a daemon module which listens for new connections from clients, and a connection module which handles connections from clients to one or more servers. The system architecture can be seen in figure 2.

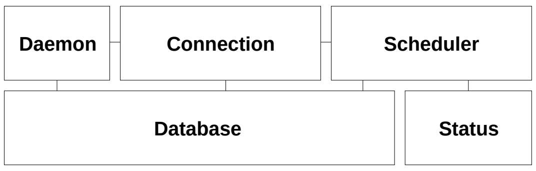

Fig. 2. The main components of the developed system and there inter-modular communication

*Process architecture* In order to gain the desired performance improvement regarding search queries, some parts of the system must run concurrently. The needed threads have been identified as being:

- A thread executing the daemon module waiting for new connections from clients.
- A thread for each currently connected client, thereby enable parallel scheduling.
- A thread for each host gathering status information.

The connection threads are to be considered the controlling threads in the system and as such the life of a connection thread can be used to illustrate all major parts of the developed system.

The system performs the following actions for a single connection:

(1) A client connects to the proxy. The daemon module detects the connection and spawns a new connection thread to handle the connection.
(2) The connection thread receives username, password and name of the database to which the client want to connect. This is validated in the scheduler module using cached information about clients. If the user is unknown a new connection to a database server is made. If the client could not be authorized the connection is terminated.
(3) A query is read from the client and examined.
    (a) If the query is read-only the scheduler is asked for a connection to the currently least busy database server with the same credentials as the client used for login. The query is then forwarded to the database server and the result returned to the client. If no errors occurred the database connection is returned to the scheduler for reuse by other queries, otherwise it is removed.
    (b) If the query modifies the database the scheduler is asked for a connection to all database servers and the affected database is locked, causing any other threads requesting to update the database to wait. A thread is spawned for each database server, and all servers are updated in parallel. After all servers have been updated, the result is returned to the client, and the database is unlocked. Like in the case of read-only queries each connection is either returned to the scheduler for reuse or discarded.
(4) Step 3 is repeated until the client terminates the connection or all database servers have crashed.

### 2.5 *Test specification*

In order to test the overall system design a proof-of-concept prototype has been made. The cluster was build using PostgreSQL 7.0.3 Database Managers (PostgreSQL, Org., 2000a), and as such large parts of the postgreSQL protocol (PostgreSQL, Org., 2000b) had to be implemented in order to mimic the interfaces. In addition to the database protocol the developed system used the UCDavis implementation of Simple Network Management Protocol (The NET-SNMP Project, 2000) to gather status information from the hosts in the cluster.

In order to determine how the developed system scales when increasing the number of database servers a performance test has been made. The test has been conducted for a varied ratio of searches and updates.

The performance test of the developed proxy has been conducted on a database containing 50000 tuples with the attributes shown in table 1.

Table 1. Attributes used in the test database.

| Name | Type | Contents |
|---|---|---|
| id | int | (current tuple number) |
| text | varchar(255) | this is row (current tuple number) of the database |

The two types of queries are:

- `select count(text) from test where text like '% N %';`
- `update test set text='this is row N of the database1' where id=N;`

Where `N` is a random integer between 0 and 49999.

The test was performed by measuring the time it takes to execute 10 clients in parallel each performing 100 queries to the database. From the result the amount of queries executed by second was calculated. The test was conducted using 0, 25, 50, 75 and 100% updates respectively.

Each of the sub-tests was performed three times and mean value was calculated. The test was performed on one database server without using the developed proxy and on one, two and three database servers when using the developed proxy.

The performance test was performed using identical computers as database servers. The computers used were equipped with dual P3-500 processors, 128 MB RAM and was placed on a 100 Mbit switched Local Area Network.

## 3. RESULTS

The results found during the test is shown in table 2.

Table 2. Test results. The values shown are the number of queries performed per second.

| Updates | Reference | 1 host | 2 hosts | 3 hosts |
|---|---|---|---|---|
| 0% | 6.2 | 6.2 | 12.2 | 18.1 |
| 25% | 6.1 | 6.0 | 6.5 | 6.3 |
| 50% | 6.1 | 6.5 | 6.3 | 6.3 |
| 75% | 6.4 | 6.3 | 5.9 | 5.5 |
| 100% | 6.2 | 6.2 | 4.8 | 4.2 |

The test-values found using the developed proxy is shown in figure 3 in relation to each other.

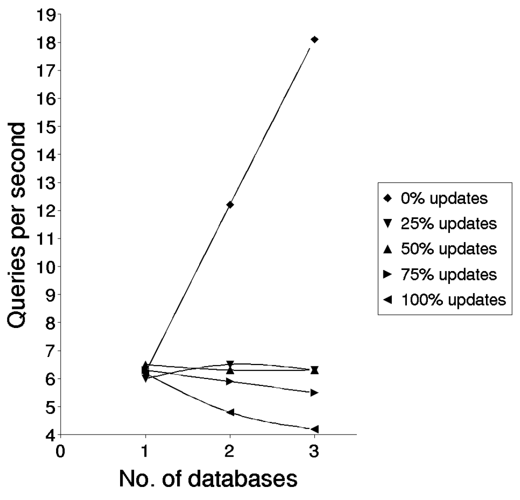

Fig. 3. Test results when using the developed proxy.

As it can be seen from figure 3 the performance of the developed proxy scales almost linearly with the number of used databases when all the queries are searches. It can also be seen that the performance drops severely when increasing the amount of performed updates. By examining the values in table 2 it can be seen that the developed proxy using up to three database serves performs better than a single site database server, when the percentage of update queries are 50% or less. It should be noted though that this break-even point would most likely change if the proxy were used in systems with significantly different queries than the two used in the performance test.

Therefore it must be concluded that it is possible to improve performance of search intensive centralized systems using techniques known from distributed systems.

## 4. DISCUSSION

This article has proven that it is possible to use methods from distributed system to improve performance and flexibility of a centralized database system. However some areas of the developed system have potential for further improvement. The two most important areas are:

- Updates of the database
- Ensuring fault tolerance
- Recovery management

At present time the database is locked on all database servers in the cluster when an update occurs. After the update is completed, the database is unlocked again. Performance can be increased by using more fine-grained locking, hence only locking the affected tables, not the entire database. This way multiple updates of the same database could take place in parallel as long as updates does not affect the same tables. Furthermore a more advanced commit protocol could be used.

In the current system a breakdown of the proxy will cause a breakdown of the entire system. Either of the two methods mentioned in section 2.2 could be used to ensure higher fault tolerance. It would be relatively simple to implement it as a primary-backup system having a proxy running on each database servers in the cluster working together as a primary-backup system. If the primary proxy fails one of the backups could be elected as the new primary, and the system would keep running. The problem with this improvement is that clients need to have some means of knowing which server currently is elected as primary. Either the clients have to be modified, hereby breaking the requirement of the system concerning that it should behave transparently towards the clients, or some external means of distribution must be used. An example of this would be a domain name server which could direct the clients to the IP address of the current primary.

At present time the developed system detects fail-stop failures within the database cluster and removes the affected server from the list of available databases. The developed system does not provide means for recovery other than manually coping the database from a non-faulty server to the affected server. System availability could be improved by utilizing one of the recovery techniques discussed in 2.3. All of the suggested recovery techniques can be implemented without major redesign of the developed system.

## References

MySQL, Inc.
WWW: http://www.mysql.com/ , 2000a.

MySQL, Inc. The todo list.
WWW: http://www.mysql.com/development/todo.html , 2000b.

F. B. S. Navin Budhiraja, Keith Marzullo and S. Toueg. The Primary-Backup Approach. In S. Mullender, editor, *Distributed Systems*, chapter 8. Addison Wesley, 1994. ISBN 0-201-62427-3.

PostgreSQL, Inc. eRServer - PostgreSQL Enterprise Replication Server.
WWW: http://www.erserver.com/ , 2000.

PostgreSQL, Org.
WWW: http://www.postgresql.org/ , 2000a.

PostgreSQL, Org. Frontend/Backend protocol used by PostgreSQL.

WWW: http://www.postgresql.org/docs/programmer/protocol.htm , 2000b.

F. B. Schneider. Repliction Management using the State-machine Approach. In S. Mullender, editor, *Distributed Systems*, chapter 7.
Addison Wesley, 1994. ISBN 0-201-62427-3.

M. D. Schroeder. A State-of-the-Art Distributed System: Computing with BOB. In S. Mullender, editor, *Distributed Systems*, chapter 1.
Addison Wesley, 1994. ISBN 0-201-62427-3.

A. Silberschatz, H. K. Korth, and S. Sudarshan. *Database Design Concepts*.
McGraw-Hill, 3. edition, 1997. ISBN 0-07-114810-8.

The NET-SNMP Project. The NET-SNMP Project Home Page.
WWW: http://net-snmp.sourceforge.net , 2000.

W. E. Weihl. Transaction-Processing Techniques. In S. Mullender, editor, *Distributed Systems*, chapter 13.
Addison Wesley, 1994. ISBN 0-201-62427-3.