ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
DML	Data Manipulation Language
DBMS	Database Management System
RAID	Redundant Array of Inexpensive Disks
RDBMS	Relational Database Management System
SaaS	Software as a Service
NIC	Network Interface Controller
TCO	Total Cost of Ownership
GPL	Gnu General Public Licence
DRBD	Distributed Replicated Block Device
OCFS	Oracle Cluster File System
ext3	third extended filesystem
GFS	Global File System
CRM	Cluster Resource Manager
LVM	Logical Volume Manager
RAM	Random Access Memory

1 INTRODUCTION

1.1 Objectives of the study

The goal of this thesis is to explore the engineering of business critical database driven web application software. The software is meant to be offered as a web service accessible with web browsers. Software being business critical here means that it has to offer high availability for the whole system and guarantee that the data users input to the system stays correct, has internal integrity and is secured against data loss.

1.2 Scope of the study

Open source software has expanded and its quality has increased so much that many of the world's largest companies offering services on the internet rely on it. Open source also offers numerous benefits for the software development process:

- Source code is shared which enables developers to directly inspect how the software works.
- With access to source code, developers can easily expand the code, building on top of it.
- Because of the copyleft licenses, developers can take it and use it freely on commercial products.
- A tested and proved open source software stack can be acquired prepackaged and be used as a leverage towards building new products in very short time and with very little man power. Reinventing the wheel is not required.

The scope of this study is to look into existing open source software as building blocks for achieving the study objectives. Commercial alternatives are also mentioned briefly, where appropriate, but the main focus is in the open source. Open source landscape is also varied enough to offer several popular projects for achieving the engineering goals.

1.3 Structure of the study

First Chapter outlines the thesis goals, scope and structure. Second Chapter details the requirements for engineering of business critical web application software. Theoretical part of this thesis is in Chapters 3-5. Third Chapter discusses options for data protection. Fourth Chapter discusses options for implementing high availability for the web service. Fifth Chapter discusses performance and scalability issues, as those are often related to high availability. Sixth Chapter details a few different strategies in engineering the system and selecting the software to fulfill the system requirements. In the seventh Chapter these systems are evaluated and example systems are built and tested against the requirements. Conclusions are represented in eight Chapter.

2 REQUIREMENTS FOR THE SYSTEM

Because business critical systems are used to take care of day-to-day business activities, requirements for software are quite complex. Businesses will be using the software to store and access important business critical data. Nature of the software will therefore present numerous challenges for the engineering of the software:

- Data integrity and safety must be preserved in all situations, including software and hardware failure.
- Data security must be preserved. Because the software will be exposed to the internet, software will be open to malicious hacking attempts.
- The system must have high availability.
- Performance issues must be addressed for future growth before they become a problem for user experience.

2.1 Data protection

Data that forms the web application typically includes the application code, database data and document and media files that are served to the user. System must be able to protect the data and ensure that the data stays intact. Application code consists of files that are read and executed on the server and results are then sent to user by the web server software. Application code can be compiled before the execution to a binary or during the execution dynamically.

Database data resides usually on the disk as files that are accessed by database, although there are databases that store the data directly on the memory. For the purposes of this thesis it is assumed that the application database stores its data on hard disk storage. Database management systems (DBMS) organize the storage of data. DBMS controls the access and maintenance of the database data. DBMS task is also to protect the data. DBMS often offers the tools to enforce data integrity, so that the data stays logically coherent during user modifications and when the system faces hardware or software failures.

File storage exists to handle large amounts of document and media files that users interact with in the application. Files in the file storage are mostly static and they are not usually modified once they are uploaded on the system until deleted. File storage can be common with application code and database data storage. Files can also be served from external location better suited for serving large amounts of static files.

2.2 Data security

Oracle (2008b) details common tasks for ensuring data security:

- Ensuring that the database installation and configuration is secure.
- Managing the security aspects of user accounts: developing secure password policies, creating and assigning roles, restricting data access to only the appropriate users etc.
- Ensuring that network connections are secure.
- Encrypting sensitive data.
- Ensuring the system has no security vulnerabilities and is protected against intruders. Administrator of the system is usually responsible for downloading and installing security patches to protect the system against vulnerabilities.

2.3 Availability of the software

Availability is the degree to which an application, service, or function is accessible on demand. Availability is measured by the perception of an application's end user. End users experience frustration when their data is unavailable or the computing system is not performing as expected. End users do not understand or care to differentiate what is causing the service disruption. For example, performance failures due to higher than expected usage create the same problems as the failure of critical hardware component.

According to Oracle (2005), highly available solution has a few primary characteristics:

- Reliability: Reliable hardware is one component of a high availability solution. Reliable software, including the database, Web servers, and applications, is just as critical to implementing a highly available solution. A related concept is resilience. For example, low-cost commodity hardware, combined with special software, can be used to create a computer cluster with redundant nodes, which are then used to provide service when system components fail. Related concept is horizontal scaling, which is detailed in Chapter 5.4.
- Recoverability: Because there may be choices for recovering from a failure, it is important to determine what types of failures may occur in a high available environment and how to recover from those failures in a timely manner that meets business requirements.
- Timely error detection: If a component in architecture fails, then fast detection is essential to recover from the unexpected failure. Monitoring the health of environment requires reliable software to view it quickly and the ability to notify administrators of a problem.
- Continuous operation: Providing the ability for continuous access to your data is essential when very little or no downtime is acceptable to perform maintenance activities. Administrative activities, such as making changes to database, or changing hardware components, should be transparent to the end user in a high availability architecture.

3 DATA PROTECTION

Most important part of any database driven web application is the data it collects from its users. Protecting this data is most important aspect of maintaining the application operational. This chapter goes through aspects of data protection.

3.1 Relational model and relational database

A database is a collection of logically related data or files. A database is designed and built to give access for applications to retrieve and modify the data. Information in a database is structured according to a database model. Database model defines the structure of the database and how to access the database. According to Lehtimäki (2009) some of the often used models are:

- Hierarchical model
- Network model
- Relational model
- Entity-relationship model
- Object-relational model

The most interesting model of these for designing Web applications is the relational model, which is implemented in many of today's most popular database management systems, including MySQL, PostgreSQL, Microsoft SQL Server and Oracle. Relational model addresses three aspects of data:

- Data structure (or objects)
- Data integrity
- Data manipulation (or operators)

All three parts have their own special terms. Terms concerning objects and their correspondence to relational databases are explained in Table 1 (Date, 1995).

Table 1. Concepts in the relational model concerning data structure and their correspondent application in relational databases (Date, 1995).

Relational model	Relational database
Domain: Pool of values, from which actual attribute values are drawn.	*Data type*: Integer (INT), character string (varchar) etc.
Attribute: An attribute is an ordered pair of attribute name and type name. Values of an attribute must come from a single domain.	*Column*: A column in a table, with name and valid data type.
Tuple: Unordered set of attribute values. The number of tuples is called the *cardinality* and the number of attributes is called the *degree*.	*Row*: A row in the table.
Relation: Relation consists of two parts, a heading and a body. Heading is a set of attributes, and body is a set of n-ary tuples.	*Table*: Heading is the row of column headings and the body is the set of data rows.

Various features of relational database make it especially useful for ensuring data integrity, like schema constraints and transactions. MySQL is used in this thesis as an example of RDBMS (Relational Database Management System) for detailing replication, which is a key technique to ensure data safety in event of system failures. The RDBMS are powering some of the most popular sites in the internet and are widely available for development and production use.

3.2 DBMS tools for data protection

Many things can go wrong as a database is queried and modified. Problems range from the keyboard entry of incorrect data to a fire in the room where the database is stored on disk. According to Garcia-Molina, et al., (2002) some of the more common failure modes:

- Erroneous data entry can occur, when user tries to enter invalid data into database, for example forgetting to enter one digit from a phone number.
- Media failures are most common, when hard drive that contains database data suddenly breaks.
- Catastrophic failures are situations in which the media holding the database is completely destroyed. Examples include explosions, fires, or vandalism at the site of the database.
- System failures are problems that cause the state of a transaction to be lost.

A modern DMBS provides a number of software mechanisms for catching those data-entry errors that are detectable. For example, the SQL standard, as well as all popular implementations of SQL, include a way for the database designer to introduce into the database schema constraints such as key constraints, foreign key constraints, and constraints on values (for example, a phone number must be 10 digits long). Additionally triggers can be used to check that the data just entered meets any constraint that the database designer believes it should satisfy (Garcia-Molina, et al., 2002). Database schema constraints are described in more detail in Chapter 3.3.

DBMS also offers the use of transactions, where queries and other DML actions are grouped into transactions. Transactions are units that must be executed atomically and in isolation from one another. Often each query or modification action is a transaction by itself. In addition, the execution of transactions must be durable, meaning that the effect of any completed transaction must be preserved even if the system fails in some way right after completion of the transaction (Garcia-Molina, et al., 2002). Transactions are covered in in more detail in Chapter 3.4.

3.3 Enforcing data integrity with schema constraints and triggers

According to Date (1995) a database consists of particular configuration of data values. Database definition includes certain integrity rules that are supposed to prevent configuration of values, a database state, which do not have correspondence in the real

world. Database can be described to be in a consistent, or in an inconsistent state. Consistent states satisfy all constraints or integrity rules of the database schema. Oracle (2008a) lists common rules or constraints used to enforce data integrity:

- Null rule: A null rule is a rule defined on a single column that allows or disallows inserts or updates of rows containing a null (the absence of a value) in that column.
- Unique column values: A unique value rule defined on a column (or set of columns) allows the insert or update of a row only if it contains a unique value in that column (or set of columns).
- Primary key values: A primary key value rule defined on a key (a column or set of columns) specifies that each row in the table can be uniquely identified by the values in the key.
- Referential integrity rules: A referential integrity rule is a rule defined on a key (a column or set of columns) in one table that guarantees that the values in that key match the values in a key in a related table (the referenced value).

According to Oracle (2008a) SQL standard defines referential integrity rules that dictate what types of data manipulation are allowed on referenced values and how these actions affect dependent values.

- RESTRICT: Disallows the update or deletion of referenced data.
- SET TO NULL: When referenced data is updated or deleted, all associated dependent data is set to a NULL value.
- SET DEFAULT: When referenced data is updated or deleted, all associated dependent data is set to the default value.
- CASCADE: When referenced data is updated, all associated dependent data is correspondingly updated. When a referenced row is deleted, all associated dependent rows are deleted.

- NO ACTION: Disallows the update or deletion of referenced data. This differs from RESTRICT in that it is checked at the end of the statement, or at the end of the transaction if the constraint is deferred.

Triggers are described by Garcia-Molina, et al., (2002) as procedures or programs that execute whenever a modification of a certain type (for example, insertion of a tuple into relation) occurs. Triggers are then used to check that the data just entered meets any constraints that the database designer believes it should satisfy.

3.4 Transactions

The processes that query and modify the database are called transactions. A transaction, like any program, executes a number of steps in sequence. Often several of the steps will modify the database. Each transaction has a state, which represents what has happened so far in the transaction. The state includes the current place in the transaction's code being executed and the values of any local variables of the transaction that will be needed later on (Garcia-Molina, et al., 2002).

Garcia-Molina, et al., (2002) states, that properly implemented transactions are commonly said to meet the "ACID test," where

- "A" stands for "atomicity," the all-or nothing execution of transactions.
- "I" stands for "isolation," the fact that each transaction must appear to be executed as if no other transaction is executing at the same time.
- "D" stands for "durability," the condition that the effect on the database of a transaction must never be lost, once the transaction has completed.
- "C" stands for "consistency," the fact that all databases have consistency constraints or expectations about relationships among data elements (for instance, a person's date of birth must not be in the future). Transactions are expected to preserve the consistency of the database.

3.4.1 Logs to protect transactions

System failures are problems that cause the state of a transaction to be lost. Typical system failures are power loss and software errors. The steps of a transaction initially occur in main memory, so a power failure will cause the contents of main memory to disappear. Similarly, a software error may overwrite part of main memory, possibly including values that were part of the state of the program. Database systems must therefore implement techniques to support the goal of resilience, which means preserving the integrity of the data when the system fails in some way. The principal technique for supporting resilience is a log, which records securely the history of database changes (Garcia-Molina, et al., 2002).

A log is a sequence of log records, each telling something about what some transaction has done. If there is a system crash, the log is consulted to reconstruct what transactions were doing when the crash occurred. Generally, to repair the effect of the crash, some transactions will have their work done again, and the new values they wrote into the database are written again. Other transactions will have their work undone, and the database restored so that it appears that they never executed. The three principal methods for logging are undo, redo, and undo/redo, named for the way(s) that they are allowed to fix the database during recovery (Garcia-Molina, et al., 2002).

Recovery is a process whereby the log is used to reconstruct what has happened to the database when there has been a failure. An important aspect of logging and recovery is avoidance of situation where the log must be examined into the distant past. Important technique that databases implement to achieve this is a technique called “checkpointing”, which limits the length of log that must be examined during recovery (Garcia-Molina, et al., 2002).

3.5 Database backup

Garcia-Molina, et al., (2002) suggest, that backups and redundant, distributed copies – will protect against a catastrophic failure, in which the media holding the database is

completely destroyed. Therefore maintaining an archive is crucial for data protection. An archive is a copy of the database that is periodically created, either fully or incrementally, and stored at a safe distance from the database itself. Schwartz, et. al. (2008), describe additional reasons for keeping backups:

- People changing their minds: If a user intentionally deletes some data and then wants it back.
- Auditing: sometimes it is important to know what the data or schema looked like at some point in the past because of lawsuits, or application bugs.
- Testing: Easy way to test the application on realistic data is to periodically refresh a development or staging server with the latest production data.

Schwartz, et. al. (2008) state, that backups are important for availability and disaster recovery. Backups need to be planned and designed from the start of system engineering so that they don't cause downtime or reduced performance. It's also extremely important to verify the backups and build solid recovery procedures. Backups are of little use, if they don't work, or recovery causes more downtime than is accepted. Schwartz, et. al. (2008) suggests developing backup and recovery procedures based on answers on these questions:

1. How much data can be lost without serious consequences? Is point-in-time recovery needed, or is acceptable to lose whatever work has happened since the last backup?
2. How fast does recovery have to be? What kind of downtime is acceptable?
3. What is needed to recover the system? Common requirements are to recover a whole server, or just specific transactions or statements.

MySQL (2008, Ch. 6.1) describe different types of database backups:

- Logical backups save information represented as logical database structure and content.
- Physical backups consist of raw copies of the directories and files that store database contents.
- Online backups taken while database server is running.

- Offline backups require stopping the database server.

3.5.1 Logical backups

MySQL (2008a, Ch. 6.1) describes logical backups slower than physical methods because the server must access database information and convert it to logical format. Output is also larger than for physical backup, particularly when saved in text format. Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This allows more options for restoring the data, restoring just a certain part of a database. Backups stored in logical format are machine independent and highly portable. Database server does not need to be taken offline for logical backups.

3.5.2 Physical backups

MySQL (2008a, Ch. 6.1) describes physical backups consisting of exact copies of database directories and files (Schwartz et. al., 2008, call them raw backups). Physical backup methods are faster than logical because they involve only file copying without conversion. Output is also more compact than for logical backup. Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity. Physical backups are portable only to other machines that have identical or similar hardware characteristics. Backups can be performed while the MySQL server is not running.

If database is running, database vendor tools are required for locking database tables so no changes can be made (although data can still be read). The underlying files used by the database are copied to another physical location using O/S commands (such as cp, scp, tar, rsync), and once the file copy is completed, the database tables are unlocked (MySQL, 2008b).

According to Schwartz et. al. (2008), physical backups are practically a must-have for large databases; they're fast, which is very important.

3.5.3 Online versus offline backups

According to MySQL (2008, Ch. 6.1), online backups can take place while the database server is running so that the database information can be obtained from the server. The backup is less intrusive to other clients, which can connect to the database server during the backup and may be able to access data depending on what operations they need to perform. Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. Offline backups take place while the server is stopped. The backup procedure is simpler because there is no possibility of interference from client activity.

MySQL (2008, Ch. 6.1) describes that a distinction similar to backups between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients will be affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

3.5.4 Snapshot backups

Some file system implementations (for example, LVM, or ZFS) enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied (MySQL, 2008a, Ch. 6.1). According to Schwartz, et. al. (2008), snapshot-capable file systems can create a consistent image of their contents at an instant in time, which can then be used to make a backup.

3.5.5 Backups from replication slave

Replication slave can be used as a backup solution, by replicating data from the master to slave, and then backing up the data slave using logical or physical backups. The slave can be paused and shut down without affecting the running operation of the master, so it is possible to produce an effective snapshot of "live" data. Other option is to put the slave in a read-only state while doing the backup. Slave can then be changed back to read/write to continue the replication (MySQL, 2008a, Ch. 16.2.1).

3.6 Using RAID to protect data against media failures

A local failure of a disk, one that changes only a bit or a few bits, can normally be detected by parity checks associated with the sectors of the disk. Garcia-Molina, et al., (2002) advices handling major failures of a disk, principally head crashes, where the entire disk becomes unreadable, by using a RAID scheme. RAID is a computer data storage scheme that can divide and replicate data among multiple hard disk drives.

4 AVAILABILITY OF THE APPLICATION

Chaurasiya, et. al. (2007) describe high-availability clusters (also known as HA clusters or fail-over clusters) as computer clusters that are implemented primarily for the purpose of improving the availability of services which the cluster provides. They operate by having redundant computers or nodes which are used to provide service when some of the nodes experience downtime. HA clustering detects hardware/software faults, and immediately restarts the application on another node without requiring administrative actions. As part of this process, clustering software may configure the node before starting the application on it. For example, appropriate file systems may need to be imported and mounted, network hardware may have to be configured, and some supporting applications may need to be started.

Users experience the disruption in availability whether or not the downtime was planned, as explained in Chapter 2.3. Scheduled maintenance can cause the same amount of disruption as hardware failure. Chaurasiya, et al., (2007) describe that with HA clusters these kind of problems can be avoided by simply switching the services to another node and temporarily removing the node needing maintenance from the cluster. Once the work is done, the machine can be brought back into the cluster.

4.1 Planning for High Availability

Implementing high availability is a process where redundancy is built into the system, systems are made to bring replacements online when something fails. Schwartz, et al., (2008) define planning process for high availability including the following points:

1. The most important principle of high availability is to find and eliminate single points of failure in the system. To identify such points, system architect can go through the application and consider the individual parts. Is any hard drive, a server, a switch or router, or the power for one rack such point? Are all the machines in one location, or are the machines intended for redundant use provided by the same company. Other common single points of failure are

reliance on services such as DNS, a single network provider, and a single power grid. Risks also include human factors, like malicious attackers or programmer mistakes that delete or corrupt data. Planning for high availability requires a balanced view of the risks. Any point in the system that isn't redundant is a single point of failure.

2. Plan over switching (or failing over) to a standby system in the event of a failure, upgrade, application modification, or scheduled maintenance. Anything that makes the application unavailable might require a fail-over plan that also identifies how fast that fail-over needs to be. Related question is how quickly system administrator can replace the failed component after a fail-over.
3. Losing data might still be possible even if application doesn't go offline for extended period of time. If server fails, data can be lost, for example the last few transactions that was written to the database log and didn't make it to a slave's relay log. Replication techniques to address this are detailed in Chapter 5.6.
4. Separating critical and noncritical parts of the application can save a lot of work and money, because it's much easier to provide redundancy and high availability for a smaller system.

4.2 Fail-over and fail-back

Schwartz, et. al., (2008) define fail-over as the process of removing a failed server and using another server instead. This is one of the most important parts of high-availability architecture. Fail-back is the reverse of fail-over. When server A fails and server B replaces it, the server A can be repaired and failed back to it. Schwartz, et. al., (2008) also state, that full fail-over solution should, at a minimum, to be able to monitor and automatically replace a server. This should ideally be transparent to the application. Load balancing need not provide this capability.

For fail-over to work, the system needs multiple copies of each hardware. Henderson (2006, 208) categorizes the spare pieces to be cold, warm, or hot copies.

- Cold spare requires setup and configuration (either physical or in software) before it can take over for the failed component. Cold spare might be used for something like a network switch.
- A warm spare is a piece of hardware that is all configured for use and just needs to be turned on either physically or in software to start using it. A typical example might be a MySQL system with a production master and a backup slave. When the master dies all traffic can be redirected to the slave, including writes, turning it into a new slave.
- Hot spare is like a warm spare, but needs no configuration to take over the failed component. The transition happens without any user intervention. For example, two load balancers might be configured in an active/passive pair. The active balancer is taking all the traffic, talking to the backup balancer via a monitoring protocol. The active load balancer fails, and the passive load balancer stops getting a heartbeat from it. It immediately knows to take over and starts receiving and processing the traffic.

Henderson (2006, p. 209) also makes a distinction between active/passive redundant pairs, and active/active pairs or clusters. An active/passive pair consists of online production device and one hot backup that is not being used. An active/active pair uses both of the devices simultaneously, moving all traffic to a single device when the other fails. These setups are used in MySQL replication, which is detailed in Chapter 5.6. Henderson (2006, p. 209) also recommends using active/active clusters, for a few reasons. First, machines that take up rack space and draw power but don't contribute anything to the running of the application seem like a waste. Second, when machines lie idle, there's more change that something will fail when they come to be used.

Fail-back functionality is not always necessary. If two redundant components are identical, for instance, either of the two components would be as good as the other, active component can be chosen arbitrarily. Once fail-over to passive node happens, it can be the new active node even after the failed node is repaired. According to Henderson (2006, p. 209) this also avoids situation called “flapping”. In some configurations, components

may appear dead due to some software issues, so fail-over happens. Once traffic is removed from the dead component, it appears to start working again, so traffic moves back to it. The traffic causes it to fail (or appear to fail) again, so traffic moves to the spare. This is called flapping, because the traffic flaps between one component and another.

4.3 High-availability clusters

High-availability clusters consist of cluster nodes that are computers providing some services. Clusters are seen outside as one system, providing only one common place of connectivity. In the context of web service clusters, this commonly includes:

- Virtual IP address, that is shared between cluster nodes and is used to communicate between the cluster and its users.
- A web server software
- A database server
- A file server

Clusters can also provide storage as a resource, using software like DRBD (see Chapter 5.8) or shared cluster file systems to replicate the storage across nodes.

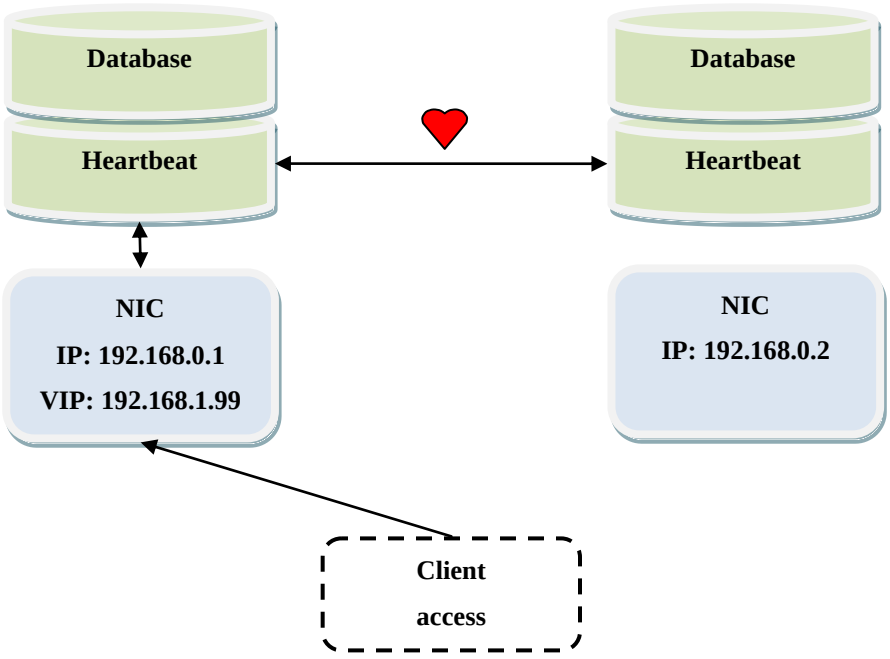

Figure 1. A high-availability cluster with a Virtual IP and MySQL database resource (LINBIT, 2008).

High-availability cluster needs a resource manager that is responsible for starting and stopping the cluster services (IP addresses, web servers, databases). Services managed by CRM (Cluster Resource Manager) are typically removed from the system startup configuration. Rather than being started at boot time, the cluster manager starts and stops them as required by the cluster configuration and status. If a machine (a physical cluster node) fails while running a particular set of services, CRM will start the failed services on another machine in the cluster, therefore providing fail-over functionality over cluster managed resources. The CRM is typically capable of automatically migrating resources back to a previously failed node, as soon as the node recovers, thus providing fail-back functionality. Figure 1 represents two-node fail-over system managed by a Heartbeat Cluster Resource Manager (Haas, et al., 2010, Ch. 9).

4.4 High-availability cluster software

Heartbeat

Heartbeat is a daemon that provides cluster infrastructure services (communication, membership) to cluster members. This allows cluster members to know about the presence or disappearance of peer processes on other machines and to easily exchange messages between them (Linux-HA, 2010).

Within heartbeat cluster each node sends a “heartbeat” signal to the other nodes in the cluster. The other cluster nodes monitor this signal. The signal can be transmitted over many different systems, including shared network devices, dedicated network interfaces and serial connections (MySQL, 2008a).

Heartbeat comes with a primitive cluster resource manager (CRM). However it is only capable of managing 2 nodes and does not detect resource-level failures. More powerful manager was spun-off from the Linux-HA project to become the Pacemaker project.

Pacemaker

Pacemaker is the successor of cluster resource manager in Heartbeat. Pacemaker project home page describes it being capable of managing clusters of practically any size (Cluster Labs, 2009).

Multi-Master Replication Manager for MySQL

Multi-Master Replication Manager for MySQL is a set of scripts that perform monitoring, fail-over, and management of master-master replication configurations. Despite its name, it can automate the fail-over process for other topologies as well, including simple master-slave and master-master configurations with one or many slaves. It uses the abstraction of a role, such as reader or writer, and a mixture of permanent and floating IP addresses (Hofmann, 2009).

Commercial software

Commercial solutions are available from Microsoft and Oracle among others that provide their own solutions for building HA clusters. For Windows platforms, Microsoft Windows 2008 Server operating system provides many high-availability features such as

Failover clustering, Network Load Balancing, Shadow Copy, Windows Server Backup, Windows Recovery and Storage Solution Availability. Microsoft (2009) claims, that because Failover Clustering is included in some editions of Windows Server, setting up such a system can be less expensive than comparable commercial solutions, which can cost thousands of dollars.

5 PERFORMANCE AND SCALABILITY

The demands for scaling and high availability often go together. High availability isn't as important when the application is small, for obvious reasons: it usually runs on a single server, so a server failure is less likely. Along with a growing number of servers, the probability of a server failing grows at the same rate. Schwartz, et al., (2008, p. 410) define scalability as an ability to add capacity as needed without reducing performance. If the application is scalable, more servers can be plugged in to handle the load, and performance problems disappear. If it's not scalable, system administrator can find himself concentrating on the performance problems, trying to tune servers, and so forth. This is focusing on symptoms, not the root cause, and can be avoided by planning for scalability.

5.1 Load balancing

When scaling system vertically, spreading load between the various processors and resources is a job of the operating system scheduler. New problems appear when scaling horizontally, because there's no operating system to spread requests between multiple nodes. Typical case for load balancing is multiple requests coming to in to the same IP address, which need to be serviced with multiple nodes.

5.1.1 DNS load balancing

According to Henderson (2006, p. 214), the easiest way to load balance between a couple of web servers is to create more than one “A” record in the DNS zone for the applications domain. DNS servers shuffle these records and send them back in a random order to each requesting host. However, adding or removing any nodes from the pool is a slow process. It could take up to a couple of days to make a change to the zone that appears for all users. For this and other reasons DNS load balancing is not a very practical solution.

5.1.2 Hardware load balancing

According to Henderson (2006, p. 214), the most straightforward way to balance requests between multiple machines in a pool is to use a hardware appliance. It is plugged in, configured and ready to start serving traffic. Compared to DNS, adding and removing real servers from the VIP (Virtual IP) happens instantly. Hardware load-balancing allows the creation of completely automated fail-over clusters. When a real server in the VIP pool dies, the load balancer can detect this automatically and stop sending traffic to it. Hardware load-balancing devices tend to be very expensive, prices starting at five figures and going up. An example of hardware load-balanced network is shown in Figure 2.

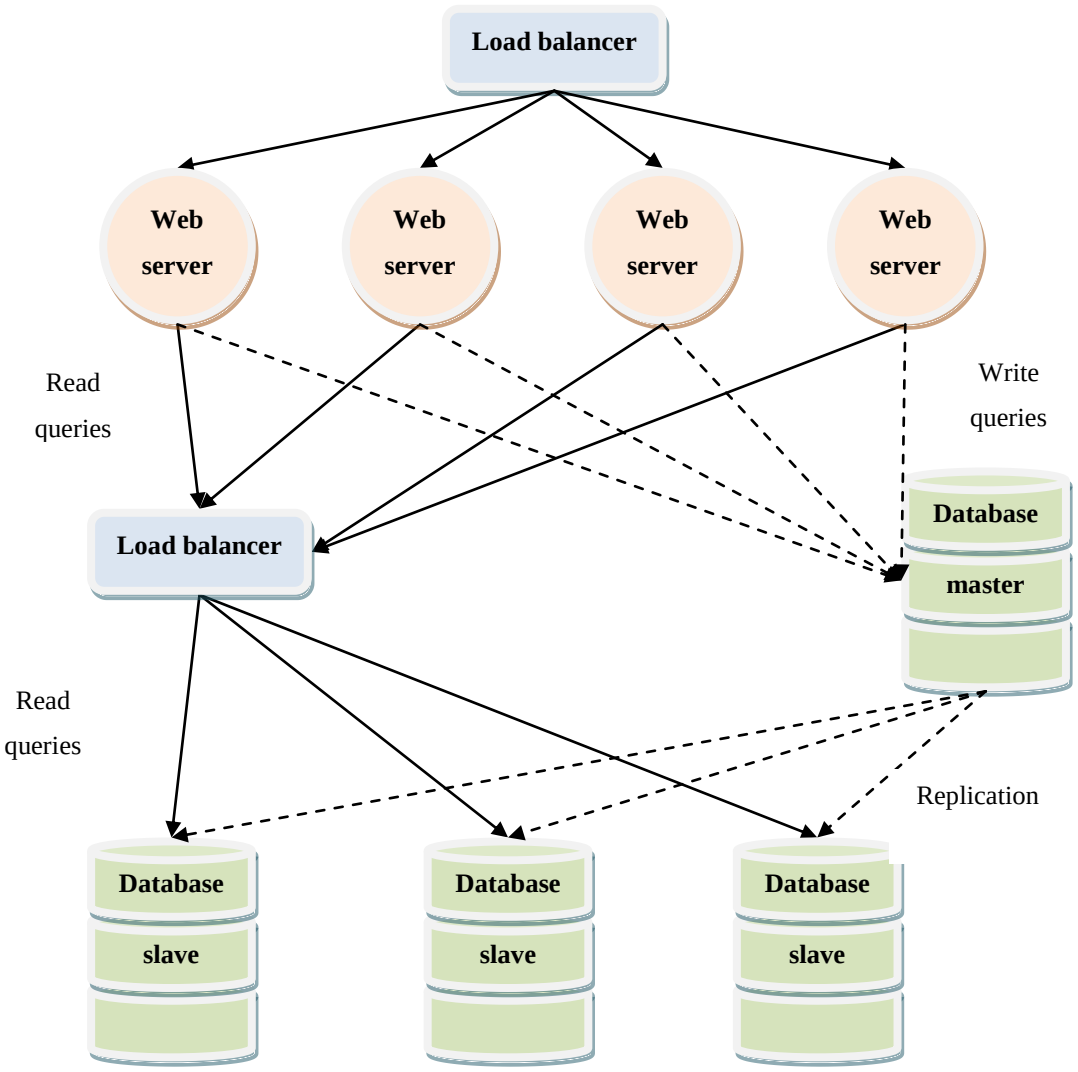

Figure 2. An example of application setup with hardware load balancing (Schwartz, et al., 2008, p. 437).

5.1.3 Software load balancing

Load balancing can be done on the software level, using server software running on a regular machine instead of a load-balancing operating system running on an ASIC. Henderson (2006, pp. 217-218) describes that software load-balancing does not typically need special hardware, and the costs are pretty low compared to hardware appliances.

5.2 Scalability planning

According to Schwartz, et al., (2008, p. 412), the hardest part of scalability planning as estimating how much load the application needs to be able to handle. This estimation should be in the order of magnitude accuracy. If it's overestimated, resources will be wasted on development, and if it's underestimated, system will be unprepared for the load.

While hardware appears expensive at the beginning of any project, as time goes on, the cost of software becomes much more expensive (up to a certain point, when the two cross back over for huge applications). Because of this, Henderson (2006, p. 204) recommends building the application to grow such that it requires little to no software work to scale; it is better to just buy and rack more hardware.

Schwartz, et al., (2008, p. 376) describe a few easy things to do before making big scaling efforts:

- **Optimize performance**: It is often possible to get significant performance improvements from relatively simple database changes, such as indexing tables correctly or using a different storage engine. Administrator can get immediate insight into database performance by analyzing the slow query log.
- **Buy more powerful hardware**: Buying more hardware works well if the application is either small or designed so it can use more hardware well. This is common for new applications, which are usually very small or reasonably well

designed. This is in contrast to older application, where buying more hardware might not work, or might be too expensive.

5.3 Vertical scaling, or scaling up

The principle of vertical scaling is simple. As each server runs out of capacity, it is replaced with a more powerful server. The benefits of vertical scaling are that it's really easy to design for. Vertical scaling might be a valid solution in early stages of application life. When the application is still in prototype stage, it's more convenient to allocate most of the resources to development efforts, and worry about scaling later.

According to Henderson (2006, p. 204), the problem with this model is that the cost doesn't scale linearly. Figure 3 shows a price comparison for increasing the number of processors, amount of RAM, and disk capacity in a system, going either the route of buying a bigger and bigger server, or buying multiple smaller servers.

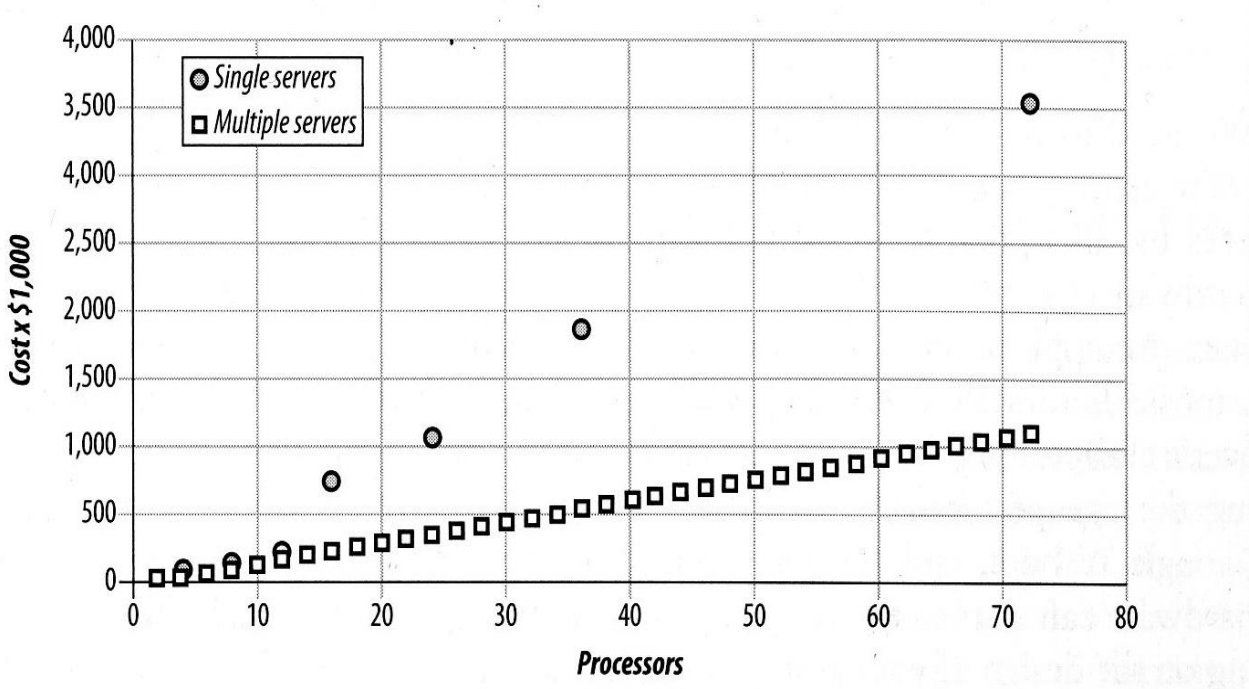

Figure 3. Costs of scaling with a small number of powerful servers vs. with a large number of small servers Henderson (2006, p. 205).

As can be seen from the Figure 3, the price growth for vertical scaling is exponential, and the available funds eventually place a limit for the scaling.

5.4 Horizontal scaling, or scaling out

The way in which horizontal scaling differs from vertical scaling is that instead of buying more powerful hardware, more regular servers are added to the system. Henderson (2006, p. 206) advices finding the TCO (Total Cost of Ownership) and computing power sweet spot, where money buys the best performance, to help plan hardware purchases. As time goes by, this price point will differ so the ideal hardware also changes over time. An application that requires identical servers is going to become expensive or impossible to scale over time, while an application that can mix and match whatever is cheapest is going to remain cheap.

According to Henderson (2006, p. 206), it is good practice to keep each node in a cluster (such as the web server cluster) the same. Otherwise the system grows more complex to manage. In a system where all nodes are identical, new boxes can be simply cloned as needed.

For horizontally scalable systems, while the hardware will scale linearly, the performance of the software running on top of it might not. For example, software that needs to aggregate results from all nodes in a cluster, or swap message among all of its peers, will not scale linearly but rather gives diminishing returns. With each new server, the amount of extra capacity decreases. Ideally software should be designed so that it always scales linearly, but this is not always practical. Henderson (2006, p. 207) recommends to monitor the situation when adding more hardware will start to get too expensive from the cost/benefit point of view. At that point it is recommended to scale vertically and replace the existing nodes in the cluster with more powerful machines. This mixture of horizontal and vertical scaling gives the benefits of both: software can be designed more easily and the most expensive servers available are not needed.

5.5 Functional partitioning

Functional partitioning is a type of horizontal scaling that Schwartz, et al., (2008, p. 376) describe as division of duties. Division can be done by dedicating individual servers or nodes to different applications, so each contains only the data its particular application needs. For example, a web site with distinct sections that don't need to share data, can be partitioned to different nodes. An example of such partitioning is presented in Figure 4.

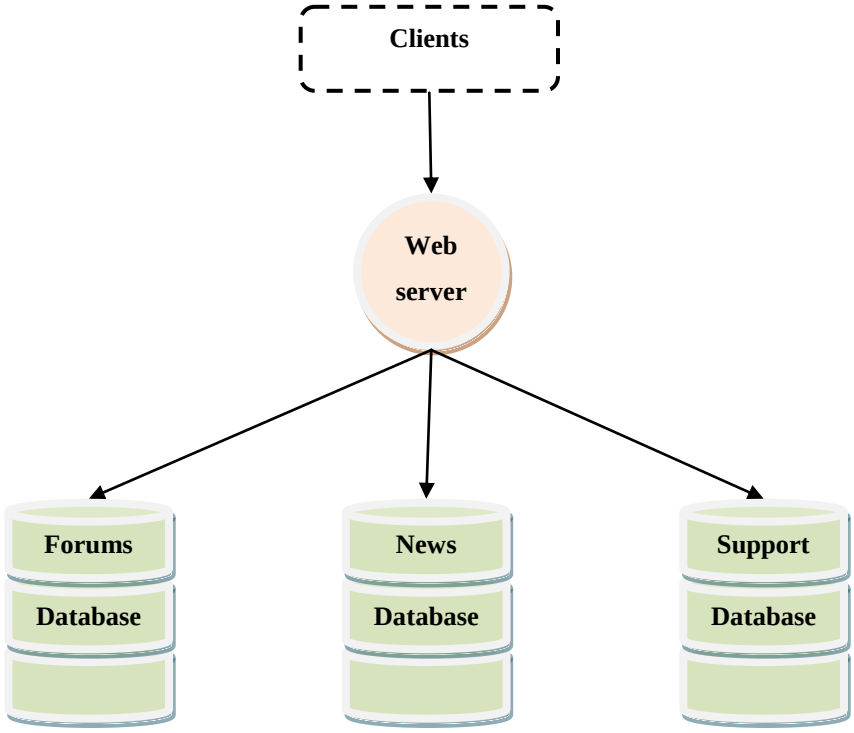

Figure 4. An example of functional partitioning for website with distinct sections (Schwartz, et al., 2008, p. 416).

Database clustering that is described in Chapter 5.7.1 is also a form of functional partitioning.

5.6 Database replication

Database replication is a form of vertical scaling. Data is replicated between multiple machines, giving us multiple points to read from. Replication has various modes, that all allow systems to expand read capacity over multiple machines by keeping multiple replicas of the table data on multiple machines.

Replication can be set up for almost any configuration of masters and slaves; however more complex modes rely on a few basic topologies. Replication is here detailed by describing MySQL replication modes. Other database products usually have the same basic topologies. Henderson (2006, pp. 232-237) describes the different basic replication modes on MySQL as master-slave replication, tree replication and master-master replication.

5.6.1 Master-slave replication

Simplest replication configuration is a single master/slave setup. All the write operations – inserts, updates, deletes and administrative commands such as creates and alters – are performed on the master. Figure 5 represents the master-slave replication setup.

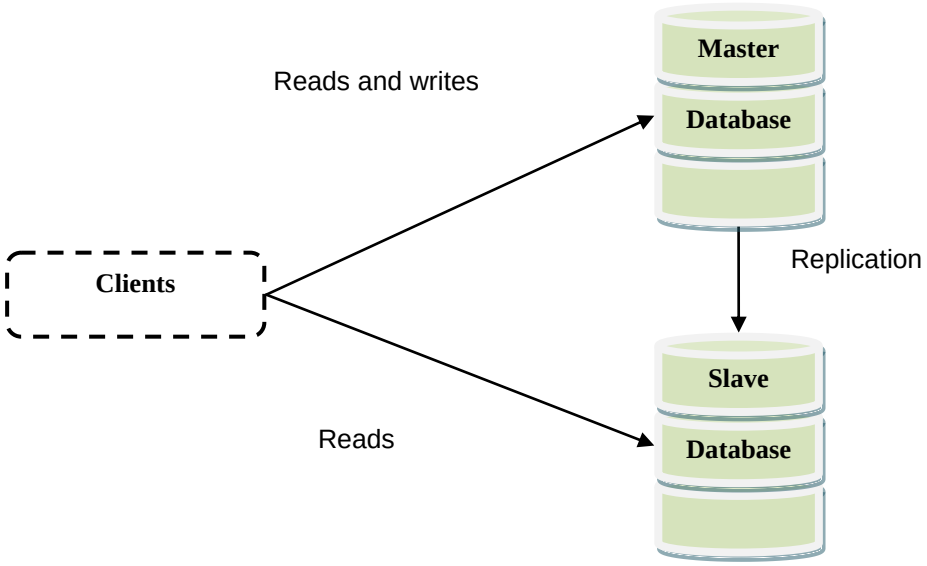

Figure 5. An example of master-slave replication sharing reads and master handling all the writes (Henderson, 2006, p. 233).

As the master completes operations it writes them to a logfile. In MySQL this logfile is called the binary log (binlog). When the slave starts up, it connects to the master and keeps the connection open. As events are written to the master's binlog, they get transmitted down to the slave, which writes them into a relay log. This connection is called the replication I/O thread at the slave end and a slave thread at the master end. The slave has a second thread always active called the slave SQL thread. This thread reads events from the relay log and executes them sequentially.

By copying the events performed on the master, the dataset on the slave get modified in exactly the same way as the master. The slave can then be used for reading data, since it has a consistent copy of the contents on the master. Any changes written to the slave do not get replicated to the master. Replication process in MySQL is shown in Figure 6.

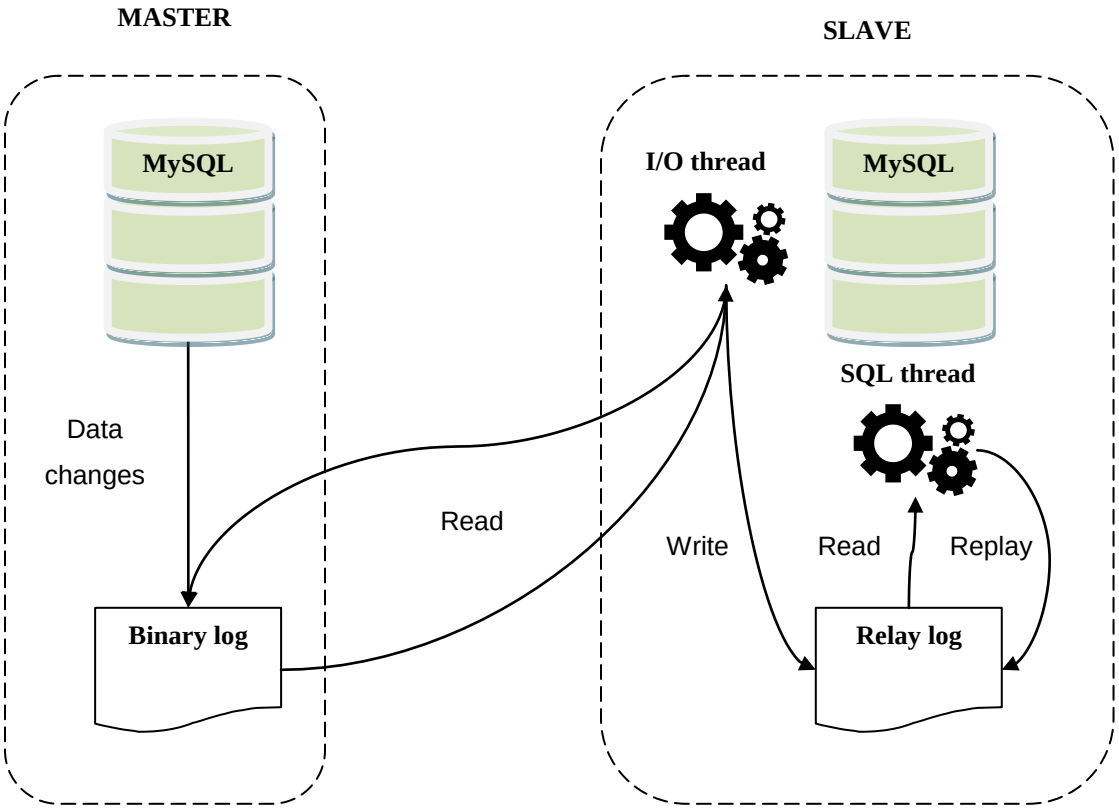

Figure 6. MySQL replication process (Schwartz, et al., 2008, p. 346).

Read capacity can be increased further by adding more slaves to the master. Each slave has its own I/O and SQL threads, relaying the binlog from the master simultaneously. Slaves aren't guaranteed to be in sync with each other as a faster machine will be able to execute writes faster than its siblings (Henderson, 2006, pp. 232-234).

5.6.2 Tree replication

Replication tree is created by turning some slaves into a master for further machines. This is done to limit the number of slaves talking to any single master. Some large web applications can have hundreds of slaves per master. The bandwidth required by the master to replicate to all slaves can become substantial because each slave requires a unicast notification of each event, all coming from a single NIC on the master. Each slave requires a thread on the master and at some point the thread management and context switching becomes a significant load (Henderson 2006, p. 234).

5.6.3 Master-master replication

Master-master replication setup has a pair of servers, each of which is a slave of the other. Each server reads the binary logs of the other, but a feature of the replication system avoids a server replaying an event that it generated. It is also possible to expand the pair of masters by considering it as a two-machine ring. Any number of servers can be added to this ring as long as each machine is a master to the machine after it and a slave of the machine before it.

Schwartz, et al., (2008, pp. 363-366) describe two modes for master-master replication, active-active mode and active-passive mode. Active-active mode involves two servers, each configured as both a master and a slave of the other. Figure 7 represents master-master replication in active-active mode.

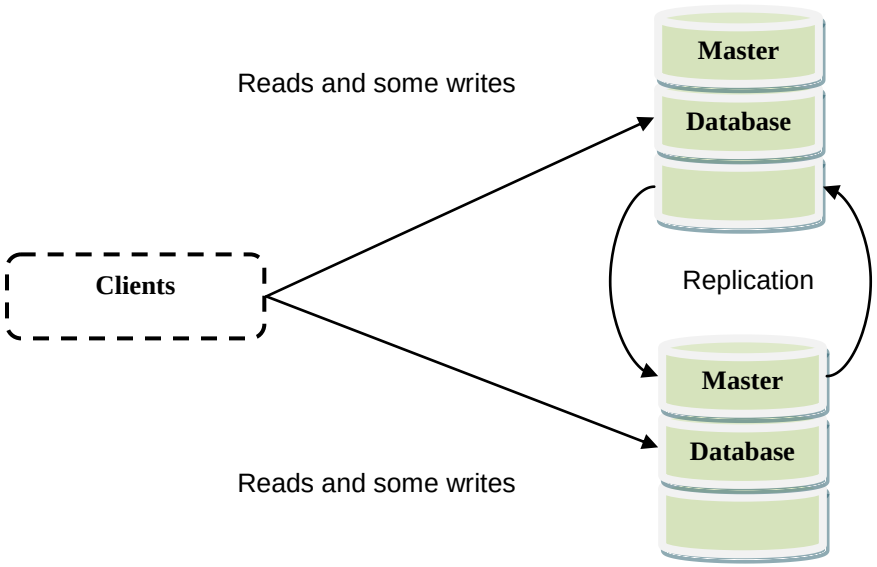

Figure 7. An example of master-master replication in active-active mode (Henderson, 2006, p. 237).

Schwartz, et al., (2008, pp. 363-366) describe few problems associated with active-active mode:

- Updates that happen in a different order on the two machines can cause the data to become out of sync silently. For example, if two writes change the same row on the same table simultaneously. Henderson (2006, p. 237) writes, that as a workaround the application code can be partitioned to make writes for some tables to server A and writes for other tables in server B.
- If replication stops with an error, but applications keep writing to both servers, both of the servers will have changes that need to be copied on the other manually.
- There's no single “true” copy of the data – at any time the data on all the machines will vary (if there's traffic), with some records existing on some portion of servers before they're replicated around. Achieving consistent reading results, for example the exact value of a row at any time, can then become a problem.

The other mode is called active-passive. Main difference to active-active is that one of the servers is a read-only “passive” server. Active-passive mode largely avoids the problems described with active-active mode. Active-passive mode allows swapping the active and passive server roles back and forth very easily, because the servers' configurations are symmetrical. This makes it easy to implement high-availability techniques fail-over and fail-back described in Chapter 4.2.

5.6.4 Ring replication

Master-master replication is actually a special form of ring replication, or multi-master replication. Rings of masters can be created with any number of servers as long as each machine is a master to the machine after it and a slave of the machine before it. Henderson (2006, p. 237) describes that one of the downsides to a multi-master approach is that there’s no single “true” copy of the data – at any time the data on all the machines will vary (assuming there’s traffic), with some records existing on some portion of servers before they’re replicated around. To get to know the exact value of a row at any time requires stopping writes to all machines, waiting for replication to catch up, and then performing the read, which is not really practical in a production system.

5.6.5 Comparison of replication topologies in MySQL

Benefits and drawbacks of different replication topologies explained in previous Chapters are represented in Table 2.

Table 2. Comparison of replication modes in MySQL (Henderson, 2006, Ch. 9).

Mode	Benefits	Drawbacks
Master-slave	• Easy way to increase read capacity almost linearly	• Transactions are not replicated as transactions. For a short time slave does not have transactional integrity while executing the writes that comprises a transaction. • Does not scale write capacity.
Tree	• Allows to limit the number of slaves talking to single master, easing the load on the master.	• If any middle-tier slave goes down, the slaves beneath it will stop replicating and go stale. • Each write event takes longer to hit the bottom-tier slaves.

Mode	Benefits	Drawbacks
Master-master in active-active mode	• Allows increasing the write and read capacity. • Increases redundancy by having no single point of failure.	• There is not single "true" copy of the data.
Master-master in active-passive mode	• Increases redundancy by having no single point of failure. • Easy implementation of fail-over and fail-back techniques.	• Very little, good all around configuration.
Multi-master or ring replication	• Allows to share read and write load between all nodes.	• There's no single true copy of the data • A failure in a single machine will cause all machines after it in the chain (with the exception of the machine just behind it) to become stale.

Schwartz, et. al., (2008) write, that since fail-back is the most important part of fail-over, symmetrical replication topologies should be preferred, such as the dual-master configuration, and asymmetrical like ring replication with three or more co-masters should be avoided. If the configuration is symmetrical, fail-over and fail-back are the

same operation in opposite directions. Fail-over and fail-back are described in more detail in Chapter 4.2.

5.6.6 Synchronous replication

MySQL replication is asynchronous. When a master fails, the most recent transactions can be lost because they might not have been copied to a slave. In synchronous replication, a transaction cannot complete on the master until it commits on one or more slave servers. MySQL does not provide synchronous replication with MySQL Server. However, MySQL also offers MySQL Cluster database, which supports synchronous replication. MySQL Cluster is a real-time open source transactional database designed for fast, always-on access to data under high throughput conditions. See Chapter 5.7.3 for more details about MySQL Cluster.

Another option is third-party provided solidDB Storage Engine for MySQL Server. solidDB Storage Engine for MySQL is a multi-threaded storage engine that supports full ACID compliance with all expected transaction isolation levels, row-level locking, and Multi-Version Concurrency Control (MVCC) with non-blocking reads and writes. solidDB is designed for mission-critical systems that require a robust, transactional database (MySQL, 2008a, Ch. 13.2).

5.7 Database partitioning

One approach to database vertical scaling is replication detailed in Chapter 5.6. However, Schwartz, et al., (2008, p. 376) suggest, that replication can help database administrator with only so far to scale the system for more reads. Slaves can split the amount of reads between themselves, but writes are also increasing at the same rate. Each slave has to write the same amount of queries in replication. This is why more and more slaves are needed to increase the read capacity by the same amount. The master-master topology described in Chapter 5.6.3 theoretically shares the writes to both nodes, but still can't handle as many writes as a single server.

To allow database to scale writes as well as reads, it must be divided into chunks. This technique is called database partitioning. Henderson (2006, Ch. 9) describes the two main techniques to do partitioning as vertical partitioning, which is also known as clustering, and horizontal partitioning also called as federation or sharding.

5.7.1 Clustering

Henderson (2006, p. 241) describes clustering as splitting a database into multiple chunks or clusters, each of which contains a subset of all tables. This typically involves a large amount of application change. By identifying which queries operate on which tables, database dispatching layer can be modified to pick the right cluster of machines depending on the tables being queried. Each cluster can be a single machine or structured according to any replication technique to suit the needs of the various tables they support. Clustering large database is shown in Figure 8.

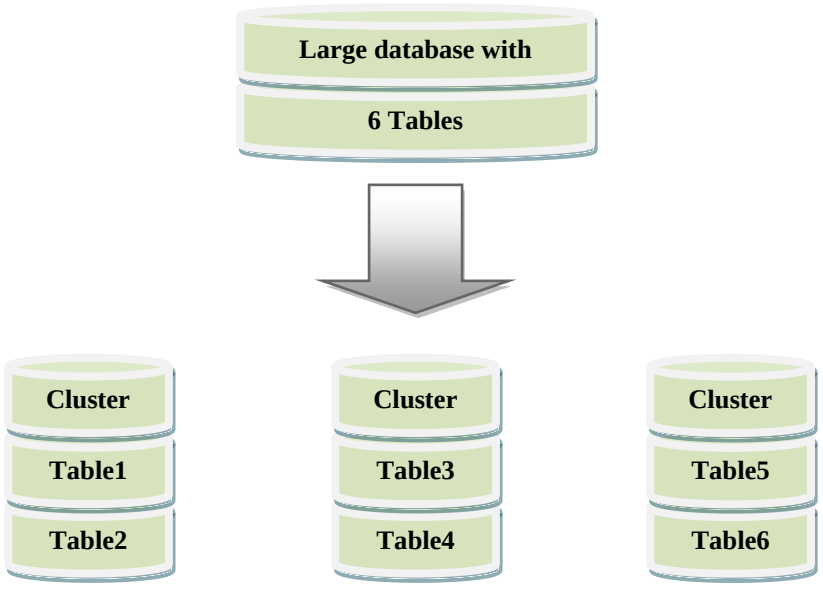

Figure 8. Database clustering (Henderson, 2006, p. 241).

Schwartz, et al., (2008, p. 412) suggest, that clustering is not a common way to partition data, because it's very difficult to do effectively and it doesn't offer any advantages over other partition methods.

5.7.2 Sharding

Slicing tables into arbitrarily sized chunks allows system administrator to add more hardware to increase both read and write capacity. This is called federation or sharding, the horizontal scaling equivalent for databases. Increasing the number of chunks can be done as the size of the dataset and the number of queries changes, always keeping the same amount of data and same query rate on each chunk. The chunks of data and the machines that power them are usually referred to as shards or cells, but are sometimes also called clusters. Figure 9 shows an example of sharding large table based on user ids.

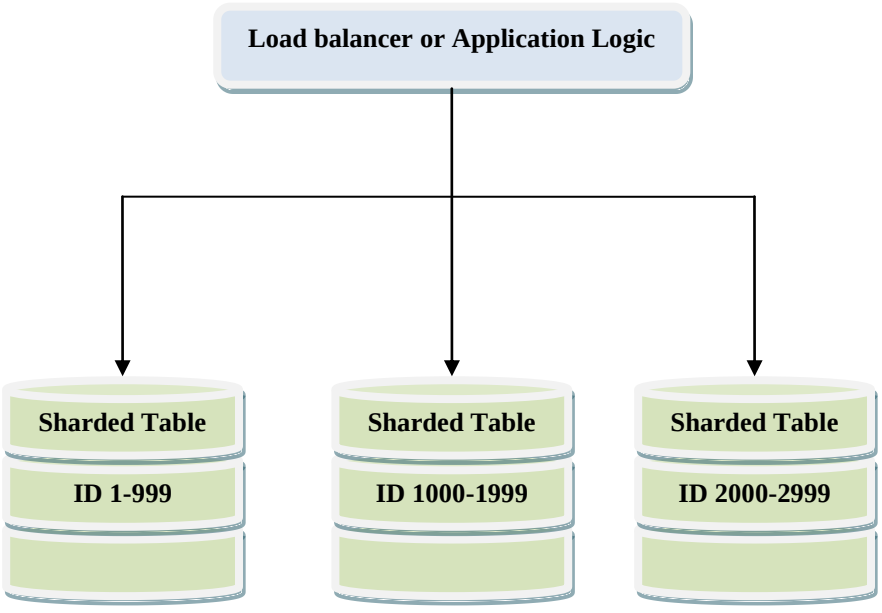

Figure 9. An example of sharding large table based on user ids (Guerrero, 2007).

Sharding can be done on the application level, meaning that application has code that handles the connections to the right shard. According to Henderson (2006, pp. 242-244) performing sharding yourself can be quite difficult compared to letting the DBMS handle

it. Selecting a range of data from a table that has been split across multiple servers becomes multiple fetches with a merge and sort operation. Join between federated tables become impossibly complicated. To address this, application data must be sharded in such as way that all the records user needs to fetch together reside on the same node. Schwartz, et al., (2008) are advising to shard only the dataset that will grow very large. As an example, a blogging service with 10 million users might not need to shard the user registration information, but if there are 500 million sharding is probably required. The user generated content, such as posts and comments, will almost certainly require sharding in either case, because these records are much larger and there are many more of them.

5.7.3 Software for database partitioning

MySQL Cluster

MySQL Cluster is separate product from MySQL AB that is based on NDB storage engine. It is a distributed, in-memory, shared-nothing storage engine with synchronous replication and automatic data partitioning with load balancing across the nodes. It is built to have no single point of failure. It can also achieve consistent response times, making it suitable for real-time applications (Sun Microsystems, 2010).

MySQL Cluster can store data either on disk or in memory. It can do this safely, because all data is distributed evenly across nodes. By storing and distributing data in a shared-nothing architecture, (that is without the use of a shared-disk) if a data node does fail, there will always be at least one additional node storing the same information. Since all data in the database is automatically replicated on multiple data nodes within the same node group, data is synchronously replicated during transactions, that is, the effect of each transaction is propagated to all the appropriate data nodes during the transaction. If a data node fails during a transaction, it is aborted and the application is informed so that it can roll-back and restart the transaction (Sun Microsystems, 2010).

Sun Microsystems (2010) describes MySQL Cluster especially suited for demanding data management solutions, like telecommunications operations and finance. MySQL Cluster works on most commodity hardware and OS platforms, requiring no special platforms or vendors. Sun Microsystems (2010) claims this results in substantially lower TCO compared to comparable commercial systems, making it a cost-effective database solution for mission critical applications. A commercial licensing and support package offering additional features for enterprise use is called MySQL Cluster Carrier Grade Edition.

memcached

According to MySQL (2008a, Ch. 14), memcached is a simple, yet highly scalable key-based cache that stores data and objects into system RAM (Random Access Memory) for a very quick access by applications.

Memcached is popularly used in combination with web application and MySQL to reduce the number of reads from the database. The application first tries to load the data from the memcached cache. If the data is not found from the cache, the database is queried and the results are written to the cache.

Advantages of using memcached according to MySQL (2008a, Ch. 14) are:

- memcached can be used to store entire objects that might normally consist of multiple table lookups and aggregations, so the speed of the application can be significantly increased because the requirement to load data directly from the database is reduced or even eliminated.
- Because the cache is entirely in RAM, the response time is very fast.
- The information can be distributed among many servers to make the best use of any spare RAM capacity.

MySQL (2008a, Ch. 14) therefore recommends memcached for high scalability situations having a very high number of reads, particularly of complex data objects that can easily be cached in the final, usable, form directly within the cache.

5.8 Block level replication

Alternative to database replication is to replicate the data in the block device level. That means mirroring a whole block device via an assigned network. Block devices are devices, which the operating system moves data in the form of blocks. In some sense this can be understood as network based RAID1. For more details about RAID, see Chapter 3.6. Implementing replicated data storage and putting the application code along the database files on the same place allows both to be replicated with the same process.

DRBD (Distributed Replicated Block Device) is software that implements block level data replication. LINBIT (2008) describes DRBD as a software-based, shared-nothing, replicated storage solution for mirroring the content of block devices (hard disks, partitions, logical volumes etc.) between servers. DRBD is originally developed by Li HA-Solutions GmbH, an Austrian company, and released to open source under GPL (Gnu General Public Licence) (LINBIT, 2008).

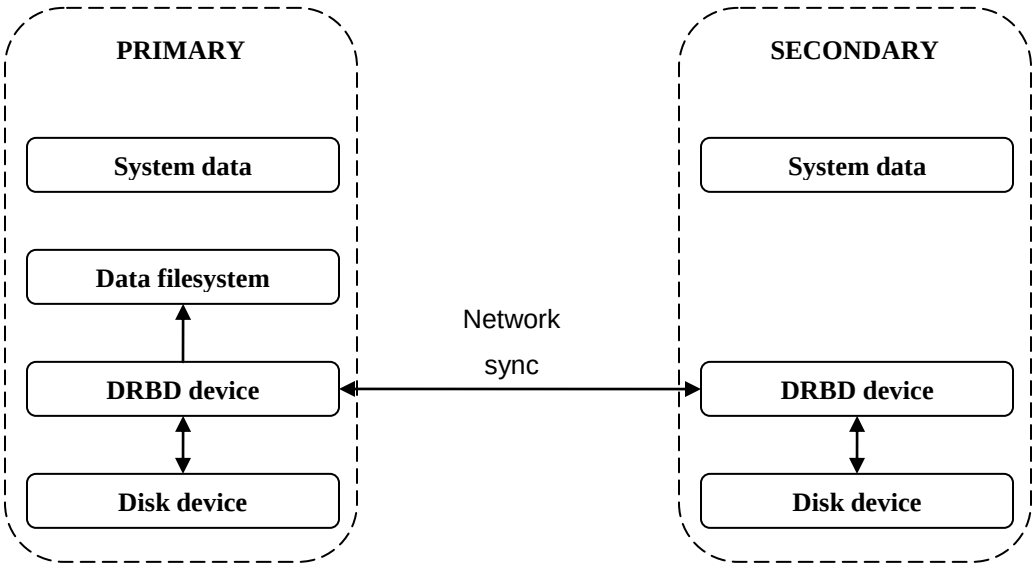

Figure 10. DRBD architecture overview (MySQL, 2008a, Ch. 14.1)

According to LINBIT (2008), DRBD's core functionality is implemented by way of a Linux kernel module. DRBD constitutes a driver for a virtual block device, so DRBD is situated at the bottom of a system's I/O stack. DRBD currently supports replication between two cluster nodes. Figure 10 represents two node DRBD cluster architecture.

5.8.1 DRBD operation modes

DRBD offers two operation modes, single-primary mode and dual-primary mode. In single-primary mode, any resource is, at any given time, in the primary role on only one cluster member. Single-primary mode guarantees that only one cluster node manipulates the data at any moment. This mode can be used with any conventional file system (ext3, XFS etc.). Deploying DRBD in single-primary mode is often used approach to implement storage for high availability clusters with fail-over capability (see Chapter 4.2) (Haas, et al., 2010).

The other mode is dual-primary mode. Dual-primary mode offers concurrent access to the the between the cluster nodes. This mode requires the use of a shared disk file system that uses a distributed lock manager. Examples include GFS (Global File System) and OCFS2. Deploying DRBD in dual-primary mode is the preferred approach for load-balancing clusters which require concurrent data access from two nodes (Haas, et al., 2010).

5.8.2 DRBD replication modes

DRBD supports three distinct replication modes, allowing three degrees of replication synchronicity. Haas, et al., (2010) describe them as follows:

- **Asynchronous replication protocol** (protocol A). Local write operations on the primary node are considered completed as soon as the local disk write has occurred, and the replication packet has been placed in the local TCP send buffer. In the event of forced fail-over, data loss may occur. The data on the standby node

is consistent after fail-over, however, the most recent updates performed prior to the crash could be lost.

- **Memory synchronous** (semi-synchronous) **replication protocol** (protocol B). Local write operations on the primary node are considered completed as soon as the local disk write has occurred, and the replication packet has reached the peer node. Normally, no writes are lost in case of forced fail-over. However, in the event of simultaneous power failure on both nodes and concurrent, irreversible desctruction of the primary's data store, the most recent writes completed on the primary may be lost.
- **Synchronous replication protocol** (protocol C). Local write operations on the primary node are considered completed only after both the local and the remote disk write have been confirmed. As a result, loss of a single node is guaranteed not to lead to any data loss. Data loss occurs only if both nodes (or their storage subsystems) are irreversibly destroyed at the same time.

The choice of replication protocol is influenced by two requirements: data protection and system latency. Asynchronous replication protocol offers the best latency, while Synchronous replication protocol offers best protection for data.

6 SOFTWARE ENGINEERING

Goal of this thesis as explained in Chapter 1 is to explore engineering options for web applications that handle business critical functions. A sample system was built for this thesis in order to be a testing ground for different engineering decisions. The system consists of:

- Operating system providing necessary background functions, like scheduled tasks
- A web server
- A database server
- Various other services, like email and monitoring

Requirements set for this kind of application in Chapter 2 guide the building of the system.

6.1 Options for data protection

In the example application MySQL was chosen as the RDBMS. MySQL is an open source database product from MySQL AB. MySQL offers these necessary RDBMS features built-in that are used to ensure data integrity described in Chapter 3:

- Multiversion concurrency through InnoDB storage engine
- Transactions through InnoDB storage engine
- Foreign key constraints through InnoDB storage engine
- Replication

The application's database schema is normalized, which means there exists multiple references between the tables. User operations sometimes involve updating data on multiple tables. InnoDB was chosen as the storage engine, because of the benefits it offers over the MySQL default storage engine, MyISAM. InnoDB was designed for transaction processing, and therefore is highly suitable for processing many short-lived transactions. This ensures, that user operations are completed in full, or, when an error

occurs, are rolled back completely without leaving orphaned data in the tables (Schwartz, et al., 2008, p. 19).

6.1.1 Database table structure

Database schema is designed to guarantee data referential integrity. Figure 11 shows part of the application database schema. For example, clients-table has a column “id”, which is 10 numbers long integer. The column also has a PRIMARY KEY index on it. Invoices table has column “client_id”, which is a FOREIGN KEY to the “id” column in clients-table. This FOREIGN KEY constraint enforces, that no invoice can be created, which doesn't reference to existing row in the clients-table.

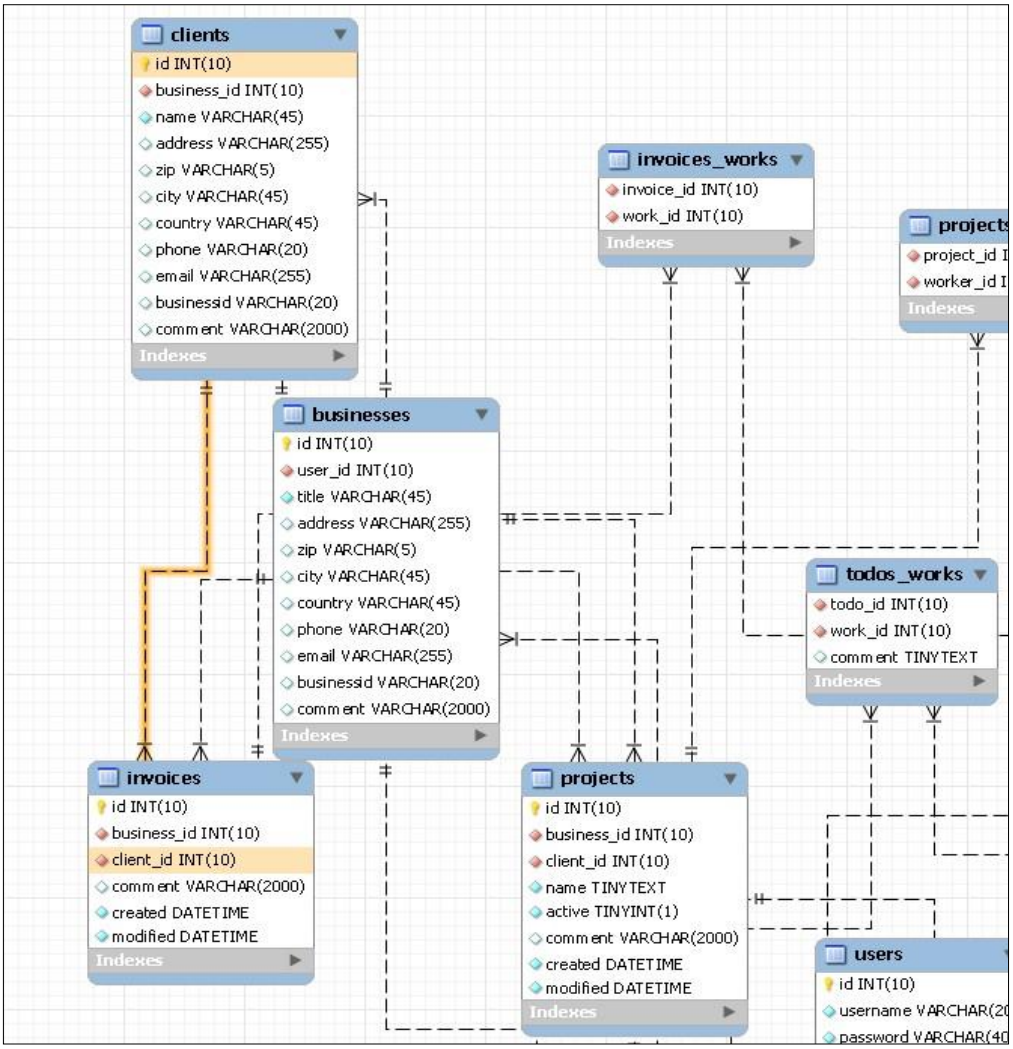

Figure 11. Screenshot from MySQL Workbench program showing a subsection of application database schema diagram.

Moreover, “client_id” column in invoices-table has a CASCADE referential constraint with “id” column in clients-table. This constraint enforces, that whenever a client row is deleted from the clients table, all referencing rows from invoices table will be deleted also. Referential constraints were described in Chapter 3.2.

6.1.2 Backing up data

Replication is not a substitute for backups and therefore it is critical that regular backups are made of database data. Table 3 describes advantages and disadvantages between various MySQL backup methods.

Table 3. Comparison of MySQL backup methods (MySQL, 2008a, Ch. 6.1 and Schwartz, et. al. 2008, Ch. 11).

Method	**Advantages**	**Disadvantages**
Backup from replication slave	• Does not disturb the master server • Allows effective snapshots of "live" data	• The slave might not have the same data as the master. Data mismatches are common, and MySQL has no way to detect this problem.
mysqldump	• Backup granularity is available at all levels, server, database or individual table. • Backups are machine independent and highly portable. • Does not require taking MySQL server offline.	• Slower than physical backup methods. • Output is larger than for physical backup methods.

Method	**Advantages**	**Disadvantages**
mysqlhotcopy (for MyISAM tables), ibbackup (for InnoDB tables), etc.	• Faster than logical because they involve only file copying without conversion. • Output is more compact than for logical backup.	• Backup granularity ranges from the level of the entire data directory down to the level of individual files. Depending on the storage engine, this may not provide table-level granularity. • Backups are portable only to other machines with similar hardware characteristics.

Method	Advantages	Disadvantages
File system snapshots (with LVM, or ZFS) with physical backups	• Taking a snapshot even for a large amount of files is a very fast operation	• File system performance storing database files can be substantially slower during the snapshot/back up process

Choosing the right backup tool depends on the requirements for backup and recovery procedures. If only 30 minutes of lost work is acceptable, database should be backed up in 30 minutes intervals. If it's important to be able to restore a database to a certain point in time, binary logs should be backed up as well with database data to be able to do point-in-time recovery.

Schwartz, et. al. (2008) suggests doing file system snapshots with LVM or ZFS and backing up the database from the snapshot as a great way to make non-disturbing online backups. Additionally, doing physical backups for large databases is preferred because of their speed advantage over logical backups. They also recommend backing up binary logs in order to enable point-in-time recovery. Snapshot backups look like a preferred method of doing backups for business critical web applications.

6.2 High availability setup

Requirements for the availability of the system were defined in the Chapter 2.3. Most important aspect for fulfilling these requirements is to find a reliable replication mechanism for database data. Two possible alternatives for data replication were

described in Chapter 3: database replication and block level replication. Table 4 details advantages, disadvantages and recommended uses between software products that use either of these two approaches.

Table 4. Comparison of replication software (MySQL, 2008a, Ch. 14).

Product	**Advantages**	**Disadvantages**
MySQL Server (database replication)	• Because replication is asynchronous, it can be started and stopped at any time. • Data can be replicated from one master to any number of slaves, making replication suitable in environments with heavy reads, but light writes (this includes many web applications), by spreading the read load across multiple servers. • Supports multiple active masters.	• Because of the asynchronous replication, there is no guarantee that data on master and slaves will be consistent at any given point in time. This can cause problems in applications where a write to the master must be available for a read on the slaves (for example a web application).

Product	**Advantages**	**Disadvantages**
MySQL Cluster (database replication)	• Offers multiple read and write nodes for data storage, allowing the write and read load to be spread across multiple servers. • Provides automatic failover between nodes. Only transaction information for the active node being used is lost in the event of a failure. • Offers synchronous replication. • Allows nodes to be added dynamically to running clusters and on-line updates to live database schema.	• Nodes within the cluster should be connected on the same high-speed LAN (Gigabit Ethernet recommended); geographically separate nodes are not supported.
DRBD in single primary mode (block level replication)	• Can guarantee data integrity across two servers in the event of hardware or system failure by using synchronous replication. • More robust design and usage than in database replication, because the block device the data is written to is replicated, not the data itself.	• Only provides a method for duplicating data across nodes. Secondary nodes cannot use the DRBD device while data is being replicated. • Cannot be used to scale performance.

Product	Advantages	Disadvantages
DRBD in dual-primary mode (block level replication)	• Data integrity is guaranteed like using DRBD with ordinary file systems. • Dual primary mode utilizes shared disk file systems (GFS, OCFS2...) to allow multiple active nodes. Can be used to scale performance.	• Shared disk filesystems are more sensitive to disturbances in the network.

According to Schwartz, et. al. (2008, Ch. 8), MySQL replication is not very resilient to crashes, power outages and corruption caused by disk, memory, or network errors. They state that manual intervention is almost certainly required at some point when replication breaks. MySQL replication can fail or get out of sync, with or without errors, just because of its inherent limitations. A fairly large list of SQL functions and programming practices won't replicate reliably.

DRBD block level replication is another choice for a small web application requiring a good fail-over solution for MySQL master node. DRBD is used in this cluster setup to enforce data integrity by using its synchronous replication protocol. DRBD combined with high-availability cluster software detailed in Chapter 4.4 then provides fail-over capabilities for the system. Part of this thesis was to find reliable replication mechanisms. A testing procedure was deviced and DRBD replication was tested against MySQL replication. Procedure and results are presented in Chapter 7.

Shared storage architecture is a way to allow multiple servers to share a single disk array. Whis this strategy, the active server mounts the filesyste. If the active server dies, the standby server can mount the same filesystem, perform any necessary recovery operations, and start operating on the failed server’s files. Shared storage architecture helps to eliminate replication errors on system crash, but introduces a single point of

failure in the system, so it's not really a viable option if the system design goal is to build redundancy (Schwartz, et. al., 2008, p. 449).

6.3 Scaling the application

High-availability requirements for the web application require building fail-over techniques for database and web server master nodes. Because most web applications are reading the database data much more than writing into it, MySQL master-slave replication can be used to scale read capacity when application grows. When the application grows even bigger, at some point database partitioning techniques detailed in Chapter 5.7 are then required to scale the system.

7 TESTING REPLICATION OPTIONS

A test to research on different database replication techniques was conducted in part of this thesis work. Testing was done on two-node MySQL master-master setup, where one of the nodes was the active and the other was passive node. Active node was also serving a web application that presented a web form. Data submitted via form was passed to active MySQL installation, and replicated to the passive node.

Goal was to test the techniques for robustness and suitability for a fail-over setup. Robustness means, that the replication does not break easily and must be able to handle various error conditions without manual intervention. Ideally, the replication setup should:

- Perform the replication reliably. Data is replicated between instances unchanged, even after a node crash when one of the nodes takes over the active role.
- Repair itself after a node crash. After the crashed node has come back online, the current active node recognizes this and starts the replication from where it was stopped without user intervention.

Techniques tested were DRBD block level replication and the MySQL's own replication. Test setups were built on Virtualbox, an x86 virtualization software by Sun Microsystems with two identical virtual machines for both setups. Both virtual machines were part of local network with private IP addressed and one Virtual IP that is assigned to the active node. Details of virtual machine and software configurations are given in Appendix I.

7.1 Test procedure

Both setups were tested with a following basic testing procedure:

- Test that the replication is working. Access the web application on primary node and input new data into database. Check that secondary node database has the same data.

- Test secondary going offline. Do an orderly shutdown on the secondary node, access the application now running on primary node and change database and turn secondary back on. Check that secondary database has the same data.
- Test primary going offline. Do an orderly shutdown on the primary node, access the application now running on secondary node and change database. Now turn primary node on. Primary and secondary node should be able to sync their data, so that primary node syncs its data from secondary, which has newer data.
- Test primary crashing. Simulate primary node crashing by pressing the reset button or similar technique. Change database on secondary and turn primary back on. Check that primary is able to sync its data from secondary.
- Load test with secondary crash. Generate load on the setup (100 serial connections by accessing and submitting the web application form) and crash the secondary node while connections are being made. After connections are made, boot secondary node up and see if the replication finishes orderly.
- Load test with primary crash. Generate load on the setup and crash the primary node while connections are being made. See if secondary takes over the connection handling (new primary) and starts handling the form submits. When primary (now new secondary) comes back online, check that replication finishes correctly between nodes.

7.2 Test results

Load tests in items 5 and 6 were made with a Selenium web application testing system running a script that made the connections to the application. Test results with brief descriptions are detailed in Tables 5 and 6. With each test item is also reported, if the replication finished correctly and if the system required manual intervention to come back fully operational. Each test item was tested several times to establish a trend.

Table 5. Replication test results for MySQL master-master replication.

Test Item	**Was the data replicated without errors?**	**Was the system able to come back online fully working without manual intervention?**
1. Replication works	Yes	Yes
2. Handles secondary orderly shutdown	Yes	Yes
3. Handles primary orderly shutdown	Yes	Yes
4. Handles primary crash	Yes	Yes
5. Load test with secondary crash	No, replication failed in each case with two identical rows being inserted into the table resulting in a broken replication.	No, out of sync tables had to be manually synchronized.
6. Load test with primary crash	Yes	Yes/No, in some cases the new primary node was reading an older binary log from the new secondary node than what was currently being written into. MySQL thought the replication was proceeding without errors.

Replication errors in item 5 occurred, because the commit to InnoDB by the slave SQL thread and the update of relay-log.info file that follows the commit are not atomic and so

transactions can get replayed. A possible subject of further study would be to test the most recent MySQL Server version to see if this problem is already addressed.

Table 6. Replication test results for DRBD block level replication setup.

Test Item	**Was the data replicated without errors?**	**Was the system able to come back online fully working without manual intervention?**
1. Replication works	Yes	Yes
2. Handles secondary orderly shutdown	Yes	Yes
3. Handles primary orderly shutdown	Yes	Yes
4. Handles primary crash	Yes	Yes
5. Load test with secondary crash	Yes	Yes
6. Load test with primary crash	Yes, sometimes DRBD lost data packets during the crash and the secondary node did not take over the primary role, but waited the crashed node to come back online and sync the data.	Yes/No, in some cases the DRBD daemons were both in a secondary role after crash recovery and data synchronization finished. DRBD serves data only from the primary role daemon.

This test suggests, that DRBD based setup is considerably more fault tolerant and requires less manual intervention to continue functioning correctly after fail-over operations. DRBD replication was able to guarantee data integrity even in case of primary node crash during write load on the setup. MySQL replication failed to guarantee

the data integrity and required manual intervention in order to continue operational after a node crash.

8 CONCLUSIONS

Building a business critical web application is possible using for the large part open source components. Combining the techniques presented in this thesis for data protection, high-availability and scaling, a consistent user experience can be achieved. Engineering of such system requires careful selection of software and hardware, good knowledge of software engineering best practices and rigorous testing procedures. Additionally, the application needs statistics collection, monitoring and alerting built into the system to maintain good user experience.

8.1 Goals of this thesis

The goal of this thesis is to explore the engineering of business critical database driven web application software. Software being business critical here means that it has to offer high availability for the whole system and guarantee that the users input data stays correct, has internal integrity and is secured against data loss. These requirements form a basis for research into several architectural and software alternatives. Requirements are detailed in Chapter 2.

8.2 Thesis research

The theory part of this thesis is in Chapters 3-5. Chapter 3 goes through key concepts and details the strategies of protecting the web application data. Chapter 4 explains the high availability concept and takes a look at various software alternatives providing the functionality. Chapter 5 goes through the concepts for splitting the data across multiple nodes from performance and data protection point of view. This Chapter also details the database replication alternatives that Chapters 6 and 7 build further upon. Chapter 6 details the decisions for engineering the software with requirements set in Chapter 2.

During the course of this thesis work, most important aspect for achieving thesis goals was to find a reliable replication mechanism for database data. A testing procedure was

deviced and DRBD replication was tested against MySQL replication. Procedure and results are presented in Chapter 7.

8.3 Thesis results

The main contribution of this study/dissertation is result that indicates database software's (MySQL) own replication mechanism inferior to more robust block level replication (DRBD) for setting up fault tolerant database master nodes. This result is expected to hold as well on other DBMS using asynchronous replication.

8.4 Subjects of further study

Replication test procedure from Chapter 7 would be interesting to conduct on several other database solutions including PostgreSQL, MySQL Cluster, Microsoft SQL Server and Oracle Database. Some of these products use synchronous replication, so their performance against DRBD block level replication would offer a good point for future research.

REFERENCES

- (Canonical, 2009) Canonical Ltd. 2009. Ubuntu Server Edition. http://www.ubuntu.com/products/whatisubuntu/serveredition, 28.08.2009.
- (Chaurasiya, et. al., 2007) Chaurasiya, V. Dhyani, P. Munot, S., 2007. Linux Highly Available (HA) Fault-Tolerant Servers. In 10th International Conference on Information Technology, (ICIT 2007). Orissa, 17-20 Dec. 2007. ISBN 0-7695-3068-0.
- (Cluster Labs, 2009) Cluster Labs. 2009. Pacemaker. http://clusterlabs.org/wiki/Main_Page, 25.01.2010.
- (Date, 1995) Date, C.J. 1995. An Introduction to Database Systems. 6th edition. Addison-Wesley Publishing Company, Inc. ISBN 0-201-82458-2.
- (Garcia-Molina, et. al., 2002) Garcia-Molina, H., Ullman, J. D. & Widom, J. 2002. Database Systems: The Complete Book. Upper Saddle River, New Jersey: Prentice Hall, Inc. ISBN 0-13-098043-9.
- (Guerrero, 2007) Guerrero, J. 2007. Scale-Out & Replication for High-Growth Businesses. MySQL Whitepaper.
- (Henderson, 2006) Henderson, C. 2006. Building Scalable Web Sites. Sebastopol, California: O'Reilly Media, Inc. ISBN 978-0-596-10235-7.
- (Linux-HA, 2010) Linux-HA. 2010. Homepage: Linux-HA. http://www.linux-ha.org/, 25.01.2010.
- (Lehtimäki, 2009) Lehtimäki, L. 2009. Course material for Databases, spring 2009. Lappeenranta University of Technology.
- (LINBIT, 2008) LINBIT. 2008. DRBD: What is DRBD. http://www.drbd.org/, 21.01.2010.
- (Haas, et. al., 2010) Haas, F., Reisner, P., Ellenberg, L. 2010. The DRBD User's Guide. http://www.drbd.org/users-guide/, 24.01.2010.
- (Microsoft, 2009) Microsoft. 2009. Windows Server 2008: Failover Clustering. http://www.microsoft.com/Windowsserver2008/en/us/failover-clustering-main.aspx, 08.10.2009.

- (Hofmann, 2009) Hofmann, P. 2009. Multi-Master Replication Manager for MySQL. http://mysql-mmm.org/, 25.01.2010.
- (MySQL, 2008a) MySQL AB. 2008. MySQL 5.1 Reference Manual. http://dev.mysql.com/doc/refman/5.1/en/index.htm, 25.01.2010
- (MySQL, 2008b) MySQL AB. 2008. Global Backup & Recovery for MySQL. http://www.mysql.com/products/backup/, 27.01.2010.
- (Oracle, 2005) Oracle. 2005. Oracle Database High Availability Overview 11g Release 1 (11.1). http://download.oracle.com/docs/cd/B28359_01/server.111/b28281/overview.htm#i1006492, 30.09.2009.
- (Oracle, 2008a) Oracle. 2008. Database Concepts. http://download.oracle.com/docs/cd/B28359_01/server.111/b28318/toc.htm, 30.09.2009.
- (Oracle, 2008b) Oracle. 2008. Database 2 Day + Security Guide. http://download.oracle.com/docs/cd/B28359_01/server.111/b28337/tdpsg_intro.htm#sthref11, 30.09.2009.
- (Pfister, 1998) Pfister, G. F. 1998. In Search of Clusters – The Ongoing Battle in Lowly Parallel Computing, 2nd edition. Upper Saddle River, New Jersey: Prentice Hall, Inc. ISBN 0-13-899709-8.
- (Schwartz, et. al., 2008) Schwartz, B., Zaitsev, P., Tkachenko, V., Zawodny, J. D., Lentz, A., Balling, D. J. 2008. High Performance MySQL, Second Edition. Sebastopol, California: O'Reilly Media, Inc. ISBN 978-0-596-10171-8.
- (Sun Microsystems, 2010) Sun Microsystems, Inc. 2010. MySQL Cluster Features. http://www.mysql.com/products/database/cluster/features.html, 27.01.2010.

APPENDICES

Appendix I: Replication test software setup

REPLICATION TEST SOFTWARE SETUP

Test setups

DRBD replication setup:

- 2 x Debian GNU/Linux 5.0 (”Lenny”)
- MySQL v. 5.0.51
- Apache2 v. 2.2.9
- Heartbeat v. 3.0 beta
- DRBD v. 8.3.2 with kernel module built for Linux 2.6.26
- One partition housing the OS and programs
- One partition managed by DRBD
- Heartbeat is the Cluster Resource Manager (CRM), handling the assigning of the Virtual IP (192.168.1.99) between nodes, mounting the DRBD partition and starting and stopping the MySQL database server and Apache2 web server
- MySQL data directory and the binary log is stored on the DRBD partition

MySQL replication setup:

- 2 x Ubuntu 8.04 (”Hardy Heron”)
- MySQL v. 5.0.51,
- Apache2 v. 2.2.8,
- Heartbeat v. 2.1.3
- Heartbeat is the Cluster Resource Manager (CRM), handling the assigning of the Virtual IP (192.168.1.99) between nodes
- Both MySQL installations are up and running, with installation on the active node receiving the traffic from the web application and the installation on the passive node replicating the data

MySQL setup details

Web application tested was using InnoDB tables to benefit from better crash recovery and data protection detailed in Chapter 6.1. Most recent MySQL Server version available through each operating system package repositories was used. MySQL replication was configured in both setups with following additional directives:

- sync_binlog=1: This makes MySQL synchronize the binary log's contents to disk each time it commits a transaction, so log events are not lost if there's a crash. Otherwise binary log entries could be corrupted or missing after a server crash (Schwartz, et. al. 2008, p. 354).
- innodb_support_xa=1: InnoDB support for two-phase commit in XA transactions is enabled, which causes an extra disk flush for transaction preparation.
- innodb_flush_log_at_trx_commit=1: Llog buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file.

DRBD setup details

DRBD replication is pretty straightforward to setup. Testing was done with C protocol (protocols are detailed in Chapter 5.8.2) and meta-disk was set on “internal”.