DATA REPLICATION STRATEGIES IN WIDE AREA DISTRIBUTED SYSTEMS

Authors:

Sushant Goel
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Ph: +613 8344 1326
Fax: +613 9348 1184
sgoel@cs.mu.oz.au

Rajkumar Buyya
Grid Computing and Distributed Systems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Ph: +613 8344 1344
Fax: +613 9348 1184
raj@cs.mu.oz.au

DATA REPLICATION STRATEGIES IN WIDE-AREA DISTRIBUTED SYSTEMS

ABSTRACT:

Effective data management in today's competitive enterprise environment is an important issue. Data is information; and information is knowledge. Hence, fast and effective access to data is very important. Replication is one such widely accepted phenomenon in distributed environment, where data is stored at more than one site for performance and reliability reasons. Applications and architecture of distributed computing has changed drastically during last decade and so has replication protocols. Different replication protocols may be suitable for different applications. In this manuscript we present a survey of replication algorithms for different distributed storage and content management systems ranging from distributed Database Management Systems, Service-oriented Data Grids, Peer-to-Peer (P2P) Systems, and Storage Area Networks. We discuss the replication algorithms of more recent architectures, Data Grids and P2P systems, in details. We briefly discuss replication in storage area network and Internet.

Keywords*: Distributed Database Management System, Data Distribution, Data Grid, Peer-to-Peer Systems, Web, and Replication.*

INTRODUCTION

Computing infrastructure and network application technologies have come a long way over the past 20 years and have become more and more detached from the underlying hardware platform on which they run. At the same time computing technologies have evolved from *monolithic* to *open* and then to *distributed* systems (Foster, 2004). Both scientific and business applications today are generating large amount of data, typical applications, such as High Energy Physics; and bioinformatics, will produce petabytes of data per year. In many cases data may be produced, or required to be accessed/shared, at geographically distributed sites. Sharing of data in a distributed environment gives rise to many design issues e.g. access permissions, consistency issues, security. Thus, effective measures for easy storage and access of such distributed data are necessary (Venugopal, 2005). One of the effective measures to access data effectively in a geographically distributed environment is replication.

Replication is one of the most widely studied phenomena in a distributed environment. Replication is a strategy in which multiple copies of some data are stored at multiple sites (Bernstein 1987). The reason for such a widespread interest is due to following facts:

(i) Increased availability

(ii) Increased performance and

(iii) Enhanced reliability

By storing the data at more than one site, if a data site fails, a system can operate using replicated data, thus increasing availability and fault tolerance. At the same time, as the data is stored at multiple sites, the request can find the data close to the site where the request originated, thus increasing the performance of the system. But the benefits of replication, of course, do not come without overheads of creating, maintaining and updating the replicas. If the application has read-only nature, replication can greatly improve the performance. But, if the

application needs to process update requests, the benefits of replication can be neutralised to some extent by the overhead of maintaining consistency among multiple replicas, as will be seen in the following sections of the paper.

A simple example of replicated environment is shown in Figure 1. Site *1*, Site *2*, Site *3*, ... Site *n*, are distributed site locations and connected through a Middleware infrastructure (for the time, it does not matter what the middleware consists of). Data stored in a file, File *X*, is stored in Site *2* and is replicated at all other sites. Suppose user *1* tries to access File *X* in Figure 1. For pedagogical simplicity, let the distance shown in the figure be proportional to the access cost of the file. The abovementioned benefits of replication are clear in this scenario (as Sites *1* and *3* are close to the user in comparison to Site *2*, where the file was originally stored). The files can be accessed at a cheaper cost (thus increasing the performance) and the file can still be accessed even if 3 out of 4 sites are down (thus increasing availability).

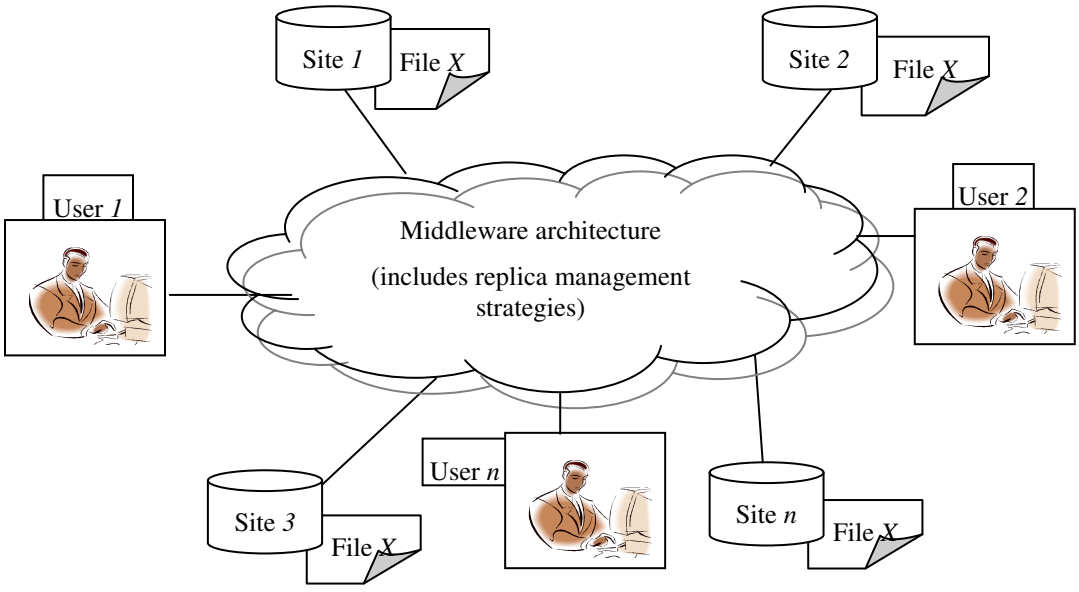

Figure 1: A simple scenario of 'File *X*' being replicated at all sites.

At the same time, continuously changing nature of computing has always created new and challenging problems in replicated environment. The purpose of this document is to review the motivation of various replication schemes in different distributed architectures. In particular, we will be covering following architectures:

(a) Distributed Database Management Systems (DBMS)

(b) Peer-to-Peer Systems (P2P)

(c) Data Grids

(d) World Wide Web (WWW)

Replication Scenario and Challenges in Replicated Environment

Various combinations of events and access scenarios of data are possible in a distributed replicated environment. For example, an application may want to download chunks of data from different replicated servers for speedy access to data; replicated data may be required to consolidate at a central server on periodic basis; data distribution on network of servers,

where some of the servers may be mobile or frequently connected (http://www.dbmsmag.com/9705d15.html); data stored at multiple sites may need to access and update the data. Based on these requirements, three types of replication scenarios can be identified:

(i) Read-only queries

(ii) Update transactions, and

(iii) Managing mobile clients.

For read-only queries, the data can be accessed by a query without worrying about the correctness of the data. As typically, the data may be generated at some site and can be read by other sites. The data can be conveniently stored at different replicated servers.

Contrary to read-only queries, update transactions need special consideration during design time. The replica management protocol may be simple if only a single site is to update the data. But, as the data can be modified by multiple sites, the consistency of the data may be compromised. To maintain the consistency of data, the order in which the transactions are executed must be maintained. One of the widely excepted correctness criterions in replicated environment is 1-copy serializability (1SR) (Bernstein, 1987). Conflicts can also be resolved with other requirements such as priority-based (a server with higher priority's update is given preference over lower priority), timestamp-based (the sequence of conflicting operations must be maintained throughout scheduling), and data partitioning (the data is partitioned and specific sites are give update rights to the partition).

Mobile computing has changed the face of computing in recent times, as well as introduced new and challenging problems in data management. In today's scenario, many employees work away from the office, interacting with clients and collecting data. Sometimes mobile devices do not have enough space to store the data, while at other times they need to access real-time data from the office. In these cases, data is downloaded on demand from the local server (http://www.dbmsmag.com/9705d15.html).

Challenges in replicated environment can be summarised as follows:

(i) Data Consistency: Maintaining data integrity and consistency in a replicated environment is of prime importance. High precision applications may require strict consistency (e.g. 1SR, as discussed above) of the updates made by transactions.

(ii) Downtime during new replica creation: If strict data consistency is to be maintained, performance is severely affected if a new replica is to be created. As sites will not be able to fulfill request due to consistency requirements.

(iii) Maintenance overhead: If the files are replicated at more then one sites, it occupies storage space and it has to be administered. Thus, there are overheads in storing multiple files.

(iv) Lower write performance: Performance of write operations can be dramatically lower in applications requiring high updates in replicated environment, because the transaction may need to update multiple copies.

CLASSIFICATION OF DISTRIBUTED STORAGE AND DATA DISTRIBUTION SYSTEMS

Considering the vast architectural differences in distributed data storage systems, we classify the data storage systems as shown in the Figure 2. The classification is based on the architectural and data management polices used in different systems. Distributed DBMSs are tightly synchronised and homogeneous in nature, while Data Grids are asynchronous and heterogenous in nature. Ownership of data is also a major issue in different systems, e.g., in peer-to-peer systems data is meant to be shared over the network, while on Data Grids the data can be application specific and can be easily shared among a particular group of people. Tightly synchronised DBMSs may store organisational specific, proprietary and extremely sensitive data. Storage Area Networks also store organisational specific data, which organisations may not want to share with other organisations. Databases in World Wide Web environment are mostly to serve client request being generated throughout the globe. Thus technologies such as Akamai (http://www.akamai.com) and disk mirroring may be a viable option in WWW environment, as the data access request is widely distributed.

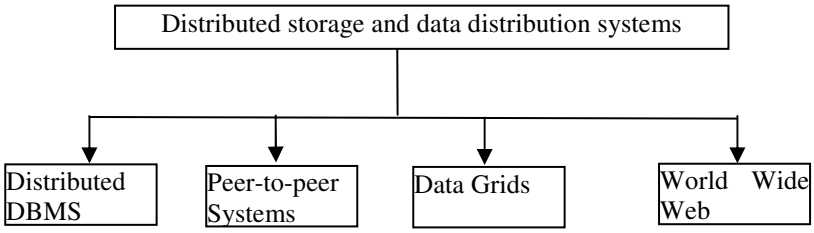

Figure 2: Classification of data storage and content management systems.

In the following sections, we will discuss the first three technologies, Distributed DBMS, peer-to-peer systems and Data Grid, in detail and we will briefly touch in SAN and WWW.

DISTRIBUTED DBMS

A replicated database is a distributed database in which multiple copies of some data items are stored at multiple sites. By storing multiple copies, the system can operate even though some sites have failed. Maintaining the correctness and consistency of data is of prime importance in a distributed DBMS. In distributed DBMS it is assumed that a replicated database should behave like a database system managing a single copy of the data. As replication is transparent from users' point of view, users may want to execute interleaved executions on a replicated database to be equivalent to a *one-copy* database, the criterion commonly known as *one-copy serializability* (1SR) (Bernstein, 1987).

Replication Protocols

ROWA and ROWA-Available

In most cases, the system is aware of which data items have replicas and where are they located. A replica control protocol is required to read and write replicated data items. The most simple replica control protocol is the Read-One-Write-All (ROWA) protocol. In ROWA protocol, a

transaction requests to read an item and the system fetches the value from the most convenient location. If a write operation is requested, the system must update all the replicas. It is clearly evident that the read operation benefits from data replication, as it can find a replica near the site of request. But, write operations may adversely affect the performance of the system. A very obvious alternative of ROWA protocol is ROWA-Available. ROWA-A was proposed to provide more flexibility to ROWA algorithm in presence of failures. Read operation of ROWA-A can be performed similar to ROWA, i.e. on any replicated copy. But to provide more flexibility, write operations are performed only on the *available* copies and it ignores any failed replicas. ROWA-A solves the availability problem, but the correctness of the data may have been compromised. After the failed site has recovered, it stores the stale value of the data. Any transaction reading that replica, reads an out-of-date copy of the replica and thus the resulting execution is not 1SR.

Quorum Based

An interesting proposal to update only a subset of replicas and still not compromise with correctness and consistency is based on quorums (Bernstein, 1987). Every copy of the replica is assigned a non-negative vote (quorum). Read and write threshold are defined for each data item. The sum of read and write threshold as well as twice of write threshold must be greater than the total vote assigned to the data. These two conditions ensure that there is always a non-null intersection between any two quorum sets. The non-null set between read quorum and write quorum guarantees to have at least one latest copy of the data item in any set of sites. This avoids the read/write and write/write conflict. The conflict table is as shown in Table 1:

Table 1: Lock compatibility matrix.

Lock held / Lock requested	*Read*	*Write*
Read	No conflict	Conflict
Write	Conflict	Conflict

All transactions must collect a read/write quorum to read/write any data item. A read/write quorum of a data is any set of copies of the data with a weight of at least read/write threshold. Quorum-based protocols maintain the consistency of data in spite of operating only on a subset of the replicated database.

Detail of majority consensus quorum protocol is shown below:

Q = Total number of votes (maximum quorum) = number of sites in the replicated system (assuming each site has equal weight)

Q_R and Q_W = Read and write quorum respectively

In order to read an item, a transaction must collect a quorum of at least Q_R votes and in order to write, it must collect a quorum of Q_W votes. The overlapping between read and write quorum makes sure that a reading transaction will at least get one up-to-date copy of the replica. The quorums must satisfy following two threshold constraints:

(i) $Q_R + Q_W > Q$ and

(ii) $Q_W + Q_W > Q$

Quorum-based replicated system may continue to operate even in the case of site or communication failure if it is successful in obtaining the quorum for the data item. Thus we see that main research focus in Distributed DBMS is in maintaining consistency of replicated data.

Types of Replication Protocols

For performance reasons, the system may either implement- (i) *synchronous* replication or (ii) *asynchronous* replication. Synchronous system updates all the replicas before the transaction commits. Updates to all replicas are treated in the same way as any other data item. Synchronous systems produce globally serializable schedules.

In asynchronous systems, only a subset of the replicas is updated. Other replicas are brought up-to-date *lazily* after the transaction commits. This operation can be triggered by the commit operation of the executing transaction or another periodically executing transaction.

Synchronous strategy is also known as *eager* replication, while asynchronous is known as *lazy* replication. Another important aspect on which the replication strategies can be classified is based on the concept of primary copy. It is represented as: (i) group and (ii) master (Gray, 1996).

(i) Group: Any site having a replica of the data item can update it. This is also referred as *update anywhere.*

(ii) Master: This approach delegates a primary copy of the replica. All other replicas are used for read-only queries. If any transaction wants to update a data item, it must do so in the master or primary copy.

A classification used in (Gray, 1996) is shown in Table 2:

Table 2: Classification of replication schemes.

	Lazy/ Asynchronous	**Eager/ synchronous**
Group	*N* transactions *N* owners	One transaction *N* owners
Master/ Primary	*N* transactions 1 owner (Primary site)	One transaction 1 owner (Primary site)

Commit Protocol in Distributed DBMS

Two-phase commit (2PC) protocol is the most widely accepted commit protocol in distributed DBMS environment that helps in achieving replica synchronisation. A 2PC protocol is defined as below:

A coordinator is typically the site where the transaction is submitted or any other site which keeps all the global information regarding the distributed transaction. Participants are all other

sites where the subtransaction of the distributed transaction is executing. Following steps are followed in a 2PC:

The coordinator sends *vote_request* to all the participating sites.

After receiving the request to vote the site responds by sending its vote, either *yes* or *no*. If the participant voted *yes*, it enters in *prepared* or *ready* state and waits for final decision from the coordinator. If the vote was *no*, the participant can abort its part of the transaction.

The coordinator collects all votes from the participants. If all votes including the coordinator's vote are *yes*, then the coordinator decides to *commit* and sends the message accordingly to all the sites. Even if, one of the votes is *no* the coordinator decides to abort the distributed transaction.

After receiving *commit* or *abort* decision from the coordinator, the participant commits or aborts accordingly from prepared state.

There are two phases (hence the name 2-phase commit) in the commit procedure: the voting phase (step (1) and step (2)); and the decision phase (step (3) and step (4)). The state diagram of 2PC is shown below:

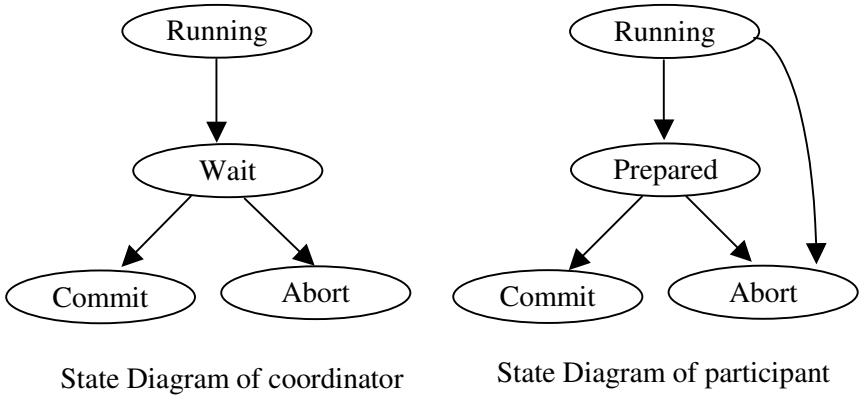

Figure 3: State diagram of coordinator and participants in 2-phase commit.

Storage Area Network

A Storage Area Network (SAN) is an interconnection network of storage devices, typically connected by high speed network (Gigabits/sec.). The storage resources may be connected to one or more servers, unlike the Direct Attached Storage (DAS) where each server has dedicated storage devices. One of the main reasons for continued interest in SAN is the flexibility of managing the distributed data. As amount of data being stored in scientific and business applications are increasing exponentially, the management and accessing of data is also getting complex. A SAN makes administering all the storage resources in such environment manageable and less complex by providing centralised data storage operations. This should not be confused with centralised storage, as is clear from the Figure 4 (A).

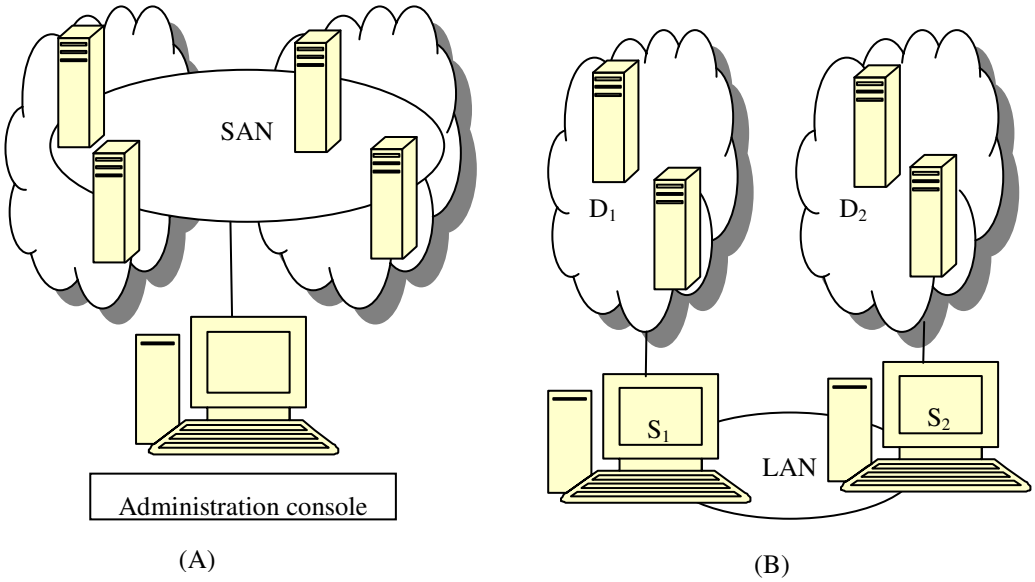

Figure 4: Schematic representation of SAN and normal distributed storage.

Figure 4 (B) shows a distributed data management architecture where administration control or servers are connected via a LAN. If a request is made at a server S_1 to access data from data cluster D_2, then the request must be routed through the LAN to access the data. The transfer speed of LAN is typically much slower than that of the SAN. SANs are connected by high-speed fibre optics cable and Gigabits/sec speed can be achieved as opposed to typical LAN speed of Megabits/sec.

SAN increases the storage performance, reliability and scalability of high-end data centres. Additional storage capacity can also be added in SAN without the need to shut down the servers. A SAN consists of various hardware components such as hubs, switches, backup devices, interconnection network (typically fibre optics cables), RAIDS (Redundant Array of Independent Disks).

SAN provides high data availability features through data replication. Two main types of replication are discussed in the literature (http://www.dothill.com/tutorial/tutorial.swf):

Storage replication: This strategy focuses on bulk data transfers. It replicates the bulk data which is independent of the application. Thus more than one application may be running on a server while the replication is being carried out to multiple servers.

Application specific replication: Application specific replication is done by the application itself and it performed by the transaction. If multiple transactions are running on the same server then the application specific replication must be used for each application.

Replication can be done either on *storage-array level* or *host level*. In array level replication, data is copied from one disk array to another. Thus array level replication is mostly homogeneous. The arrays are linked by a dedicated channel. Host level replication is independent of the disk array used. Since arrays used in different hosts can be different, host level replication has to deal with heterogeneity. Host level replication uses the TCP/IP protocol for data transfer. The replication in SAN also can be divided in two main categories based on mode of replication: (i) synchronous and (ii) asynchronous, as discussed earlier.

Survey of Distributed Data Storage Systems and Replication Strategies Used

Table 3: Survey of replication methods in different storage systems.

Attributes / **Systems**	**Type of system**	**Replication method**
Arjuna (Parrington, 1995)	Object-oriented programming system	Default: Primary copy. But simultaneous update also possible
Coda (Kistler, 1992)	Distributed file system	Variants of ROWA
Deceit (Siegel, 1990)	Distributed file system	File replication with concurrent read and writes. Updates use a write-token. Thus write update is synchronous, as the site having the token can update the data. Also provides multi-version control.
Harp (Liskov, 1991)	Distributed File System	Primary copy replication. Uses 2-phase protocol for update. Communicates with backup servers before replying to the request.
Mariposa (Sidell, 1996)	Distributed Database Management System	Asynchronous replication. Updates are propagated within a fixed time limit. (Thus having stale data among replicas)
Oracle (http://www.nyoug.org/ 200212baumgartel.pdf)	(Distributed) Database Management System	Provides basic and advanced replication. Basic: replicated copies are read only. Advanced: Replica copies are updateable. Various conflict resolution options can be specified such as latest timestamp, site priority etc. Provides synchronous and asynchronous replication
Pegasus	Distributed Object Oriented	Supports global consistency

(http://www.bkent.net/ Doc/usobpeg.htm)	DBMS (supports heterogeneous data)	(synchronous)
Sybase (http://www.faqs.org/faqs/ databases/sybase-faq/part3/	Distributed DBMS (supports heterogeneity)	Asynchronous replications

Brief explanation of systems in Table 3 is as follows. Arjuna (Parrington, 1995) supports both active and passive replication. Passive replication is like primary copy and all updates are redirected to the primary copy. The updates can be propagated after the transaction has committed. In active replication, mutual consistency is maintained and the replicated object can be accessed at any site.

Coda (Kistler, 1992) is a network/distributed file system. A group of servers can fulfill the client's read request. Updates are generally applied to all participating servers. Thus it uses a ROWA protocol. The motivation behind using this concept was to increase availability, so that if one server fails, other servers can takeover and the request can be satisfied without the client's knowledge.

Deceit (Siegel, 1990) distributed file system is implemented on top of the Isis (Birman, 1987) distributed system. It provides full Network File System (NFS) capability with concurrent read and writes. It uses write-tokens and stability notification to control file replicas (Siegel, 1990). Deceit provides variable file semantics which offers a range of consistency guarantees (from no consistency to semantics consistency). But, the main focus of Deceit is not on consistency, but to provide variable file semantics in a replicated NFS server (Triantafillou, 1997).

Harp (Liskov, 1991) uses a primary-copy replica protocol. Harp is a server protocol and there is no support for client caching (Triantafillou, 1997). In Harp, file systems are divided into groups and each group has its own primary and secondary. For each group, a primary, a set of secondary and a set of sites as witness are designated. If the primary site is unavailable, a primary is chosen from secondary sites. If enough sites are not available from primary and secondary sites, a witness is promoted to act as secondary site. The data from such witness are backed-up in tapes, so that if they are the only surviving sites, then the data can be retrieved. Read/ write operations follow typical ROWA protocol.

Mariposa (Sidell, 1996) was designed at the University of California (Berkley) in 1993-94. Basic design principle behind the design of Mariposa was scalability of distributed data servers (up to 10,000) and local autonomy of sites. Mariposa implements asynchronous replica control protocol, thus distributed data may be stale at certain sites. The updates are propagated to other replicas within a time limit. Thus it could be implemented in systems where applications can afford stale data within a specified time-window. Mariposa uses an economic approach in replica management, where a site buys a copy from another site and negotiates to pay for update streams)Sidell, 1996).

Oracle (http://www.nyoug.org/200212baumgartel.pdf) is a successful commercial company, which provides data management solutions. Oracle provides wide range of replication solutions. It supports basic and advanced replication. Basic replication supports read-only queries, while advanced replication supports update operations. Advanced replication supports synchronous and asynchronous replication for update request. It uses 2PC for synchronous replication. 2PC

ensures that all cohorts of the distributed transaction completes successfully (or roll back the completed part of the transaction.

Pegasus (http://www.bkent.net/Doc/usobpeg.htm) is an Object Oriented DBMS designed to support multiple heterogeneous data sources. It supports Object SQL. Pegasus maps heterogeneous object model to common Pegasus Object Model. Pegasus supports global consistency in replicated environment as well as also respects the integrity constraints. Thus Pegasus supports synchronous replication.

Sybase (http://www.faqs.org/faqs/databases/sybase-faq/part3/) implements a Sybase Replication Server to implement replication. Sybase supports replication of stored procedure calls. It implements replication at transaction-level and not at table-level (Helal, 1996). Only the rows affected by a transaction at the primary site are replicated to remote sites. Log Transfer Manager (LTM) passes the changed records to the local Replication Server. The local replication server then communicates the changes to the appropriate distributed Replication Servers. Changes can then be applied to the replicated rows. The Replication Server ensures that all transactions are executed in correct order to maintain the consistency of data. Sybase mainly implements asynchronous replication. To implement synchronous replication, the user should add their own code and a 2PC protocol (http://www.dbmsmag.com/9705d15.html).

PEER-TO-PEER SYSTEMS (P2P)

P2P networks are a type of overlay network that uses the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively few servers (Oram, 2001). The word peer-to-peer reflects the fact that all participants have equal capability and is treated equally, unlike client-server model where clients and servers have different capabilities. Some P2P networks use client-server model for certain functions (e.g. Napster (Oram, 2001) uses client-server model for searching). Those networks that uses P2P model for all the functions, e.g. Gnutella (Oram, 2001), are referred as pure P2P systems. A brief classification of P2P systems is shown below.

Types of Peer-to-Peer Systems

Today P2P systems produce large share of Internet traffic. P2P system relies on computing power and bandwidths of participants rather than relying on central servers. Each host has a set of “neighbour”.

P2P systems are classified into 2 categories:

(a) Centralised P2P Systems: have central directory server, where the users submit requests e.g. Napster (Oram, 2001). Centralised P2P systems store a central directory, which keep information regarding files location at different peers. After the files are located the peers communicate among themselves. Clearly centralised systems have the problem of single point of failure and they scale poorly where number of client ranges in millions.

(b) Decentralised P2P Systems: Decentralised P2P systems do not have any central server. Hosts form an ad hoc network among themselves on top of the existing Internet infrastructure, which is known as the Overlay Network. Based on two factors – (i) Network Topology and (ii) File Location, decentralised P2P systems are classified into following two catagories:

(i) Structured decentralised: Structures architecture refers that network topology is tightly controlled and the file locations are such that they are easier to find (i.e. not at random locations). Structured architecture can also be classified into two categories: (i) loosely structured and (ii) highly structured. Loosely structured systems place the file based on some hints, e.g. Freenet (Oram, 2001). In highly structured systems the file locations are precisely determined with help of techniques such as hash-tables.

(ii) Unstructured: Unstructured systems do not have any control over the network topology or placement of the files over the network. Example of such systems includes Gnutella (Oram, 2001), KaZaA (Oram, 2001) etc. Since there is no structure, to locate a file, a node queries its neighbours. Flooding is the most common query method used in such unstructured environment. Gnutella uses flooding method to query.

In *unstructured* systems, since P2P network topology is unrelated to the location of data, the set of nodes receiving a particular query is unrelated to the content of the query. The most general P2P architecture is the decentralised-unstructured architecture.

Main research in P2P systems have focused in architectural issues, search techniques, legal issues etc. Very limited literature is available for unstructured P2P systems. Replication in unstructured P2P systems can improve the performance of the system as the desired data can be found near the requested node. Specially, in flooding algorithms, reducing the search even by one hop can drastically reduce number of messages in the system. Table 4 shows different P2P systems.

Table 4: Examples of different types of P2P systems.

Type	**Example**
Centralised	Napster
Decentralised structured	Freenet (loosely structured) DHT (highly structured) FatTrack eDonkey
Decentralised unstructured	Gnutella

A challenging problem in unstructured P2P systems is that the network topology is independent of the data location. Thus, the nodes receiving queries can be completely unrelated to the content of the query. Consequently, the receiving nodes also do not have any idea of where to forward the request for fast locating the data. To minimise the number of hops before the data is found, data can be proactively replicated at more than one sites.

Replication Strategies in P2P Systems

Based on size of files (Granularity)

(i) *Full file replication*: "full" files are replicated at multiple peers based upon which node downloads the file. This strategy is used in Gnutella. This strategy is simple to

implement. However, replicating larger files at one single file can be cumbersome in space and time (Bhagwan, 2002).

(ii) *Block level replication*: divides each file into an ordered sequence of fixed size blocks. This is also advantageous if a single peer cannot store whole file. Block level replication is used by eDonkey. A limitation of block level replication is that during file downloading it is required that enough peers are available to assemble and reconstruct the whole file. Even if a single block is unavailable, the file cannot be reconstructed. To overcome this problem, Erasure Codes (EC), such as Reed-Solomon (Pless, 1998) is used.

(iii) *Erasure Codes replication*: Provides the capability that the original files can be constructed from less number of available blocks. For example, k original blocks can be reconstructed from l (l is close to k) coded blocks taken from a set of ek (e is a small constant) coded blocks (Bhagwan, 2002). In Reed-Solomon codes, the source data is passed through a data encoder, which adds redundant bits (parity) to the pieces of data. After the pieces are retrieved later, they are sent through a decoder process. The decoder attempts to recover the original data even if some blocks are missing. Adding EC in block-level replication can improve the availability of the files because it can tolerate unavailability of certain blocks.

Based on replica distribution

Following definitions need to be defined:

Consider each file is replicated on r_i nodes.

Let, the total number of files (including replicas) in the network be denoted as R (Cohen, 2002).

$$R = \sum_{i=1}^{m} r_i \text{ , where } m \text{ is the number of individual files/ objects}$$

(i) *Uniform*: The uniform replication strategy replicates everything equally. Thus from the above equation, replication distribution for uniform strategy can be defined as follows:

$$r_i = R / m$$

(ii) *Proportional*: The number of replicas is proportional to their popularity. Thus, if a data item is popular it has more chances of finding the data close to the site where query was submitted.

$$r_i \propto q_i$$

where, q_i = relative popularity of the file/ object (in terms of number of queries issued for i^{th} file).

$$\sum_{i=1}^{m} q_i = 1$$

If all objects were equally popular, then

$q_i = 1/m$

But, results have shown that object popularity show a Zipf-like distribution in systems such Napster and Gnutella. Thus the query distribution is as follows:

$q_i \propto 1/ i^{\alpha}$, where α is close to unity.

(iii) *Square-root*: The number of replicas of a file *i* is proportional to the square-root of query distribution, q_i.

$$r_i \propto \sqrt{q}_i$$

The necessity of Square-root replication is clear from the following discussion.

Uniform and Proportional strategies have been shown to have same search space as follows:

m: number of files

n: number of sites

r_i: number of replicas for i^{th} file

R = Total number of files

The average search size for file *i*, $A_i = \frac{n}{r_i}$

Hence, the overall average search size, $A = \sum_i q_i A_i$

Assuming average number of files per site, $\mu = \frac{R}{n}$

Following the above equations, the average search size for uniform replication strategy is as follows,

Since, $r_i = R / m$

$A = \sum q_i \frac{n}{r_i}$ (replacing the value of A_i)

$A = \sum q_i \frac{n\ m}{R}$

$A = \frac{m}{\mu}$ (as, $\sum_{i=1}^{m} q_i = 1$) …… …… (1)

The average search size for proportional replication strategy is as follows,

Since, $r_i = R\, q_i$ (as, $r_i \propto q_i$, and $q_i = 1$)

$$A = \sum q_i \frac{n}{r_i} \text{ (replacing the value of } A_i\text{)}$$

$$A = \sum q_i \frac{n}{Rq_i}$$

$$A = \frac{m}{\mu} \text{ (as, } \sum_{i=1}^{m} q_i = 1,\ \frac{n}{R} = \frac{1}{\mu} \text{ and } \frac{1}{q_i} = m \text{ for proportional replication) ....(2)}$$

It is clear from equation (1) and (2) that the average search size is same in uniform and proportional replication strategies.

It has also been shown in the literature (Cohen 2002) that the average search size is minimum under following condition:

$$A_{optimal} = \frac{1}{\mu}(\sum \sqrt{q_i})^2$$

This is known as square-root replication.

Based on replica creation strategy

(i) Owner replication: The object is replicated only at the requester node once the file is found. E.g. Gnutella (Oram 2001) uses owner replication.

(ii) Path replication: The file is replicated at all the nodes along the path through which the request is satisfied. E.g. Freenet uses path replication.

(iii) Random replication: Random replication algorithm creates the same number of replicas as of path replication. But, it distributes the replicas in a random order rather than following the topological order. It has been shown in (Lv 2002) that factor of improvement in path replication is close to 3 and in random replication the improvement factor is approximately 4.

The following tree summarizes the classification of replication schemes in P2P systems, as discussed above:

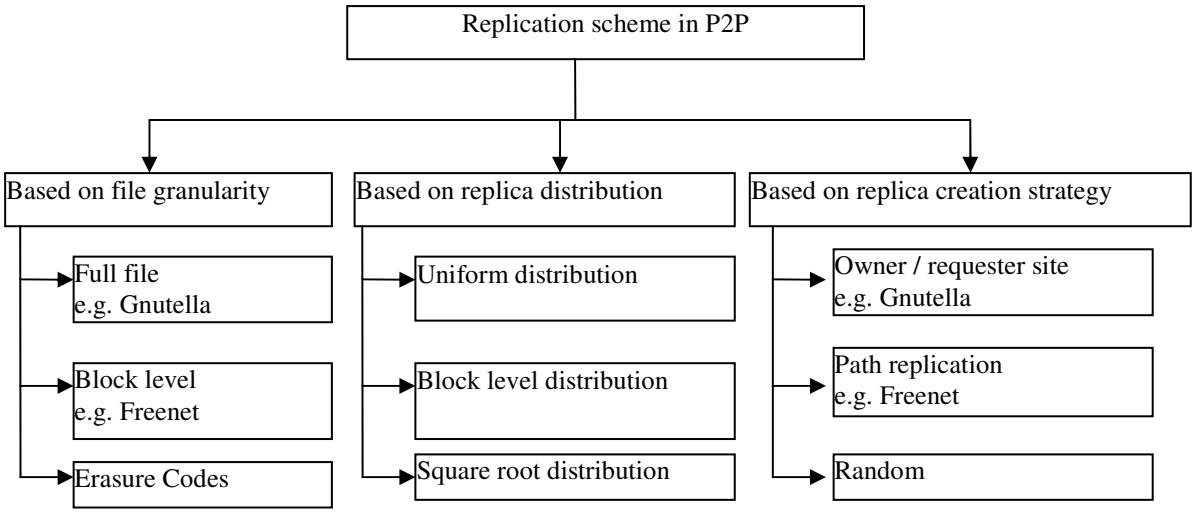

Figure 5: Classification of replication schemes in P2P systems.

DATA GRIDS / GRID COMPUTING

The amount of data being gathered and stored by advanced applications in the data sources is increasing exponentially. "The Grid" is a heterogeneous collaboration of resources and thus will contain diverse range of data resources. Heterogeneity in a Data Grid can be due to data model, transaction model, storage systems or data types. Data Grids will provide seamless access to geographically distributed data sources storing terabytes to petabytes of data.

Major assumption in Data Grids is that the data is read-only. The main motivating application for most of the research has been the particle physics accelerator. LHC is expected to produce 15 petabytes of data each year. More than 150 organisations around the world will be participating in the physics experiment. The LHC model assumes a tired architecture to distribute the data to processing sites. All data is produced at the central CERN computer center at Tier-0. The data is then distributed to Regional Centers (e.g. Asia Pacific region, USA region, EU Region etc.) at Tier-1. Regional Centers further distribute the data to Tier-2 centers, which in turn distributes the data to processing institutes at Tier-3. The data is then finally distributed to the end users at Tier-4. The rate of data transfer from Tier-0 to Tier-1, Tier-1 to Tier-2 and Tier-2 to Tier-3 is $\approx$ 622 Mb/sec. Data transfer rate between Tier-3 and 4 ranges from 10-100 Mb/sec.

As the application domain of Grid computing, and consequently Data Grid, is expanding, the architectural requirements are also changing. Thus, Data Grid needs to address architectures different than read-only environment. Hence, this is the major criterion on which we classify the replication strategies in Data Grids.

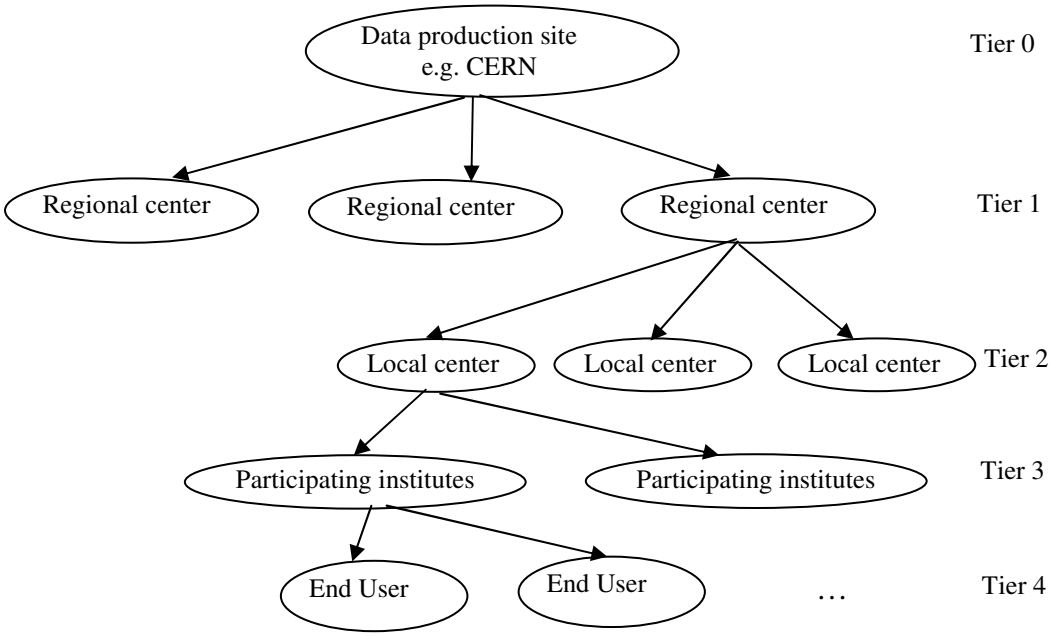

Figure 6: A tiered or hierarchical architecture of Data Grid for Particle Physics accelerator at CERN.

Replication strategy for Read-only requests

Replica Selection based on Replica location/ User preference

The replicas are selected based on users' preference and replica location. (Vazhkudai, 2001) proposes a strategy that uses Condor's ClassAds (Classified Advertisements) (Raman, 1998) to rank the sites suitability in storage context. The application requiring access to a file presents its requirement to the broker in form of ClassAds. The broker then does the *search*, *match* and *access* of the file that matches the requirements published in the ClassAds.

Dynamic replica creation strategies discussed in (Ranganathan, 2001) are as follows:

(a) Best Client: Each node maintains the record of access history for each replica, i.e. which data item is being accessed by which site. If the access frequency of a replica exceeds a threshold, a replica is created at the requester site.

(b) Cascading replication: This strategy can be used in the tired architecture discussed above. Instead of replicating the data at the 'best client', the replica is created at the next level on the path of the best client. This strategy evenly distributes the storage space as well as other lower level sites have close proximity to the replica.

(c) Fast Spread: Fast spread replicates the file in each node along the path of the best client. This is similar to path replication in P2P systems.

Since the storage space is limited, there must be an efficient method to delete the files from the sites. The replacement strategy proposed in (Ranganathan, 2001) deletes the most unpopular files, once the storage space of the node is exhausted. The age of the file at the node is also considered to decide the unpopularity of file.

Economy-based replication policies

The basic principle behind economy-based polices are to use the socio-economic concepts of emergent marketplace behaviour, where local optimisation leads to global optimisation. This could be thought as an auction, where each site tries to *buy* a data item to create the replica at its own node and can generate revenue in future by *selling* them to other interested nodes. Various economy-based protocols such as (Carman, 2002) (Bell, 2003) have been proposed, which dynamically replicates and deletes the files based on the future return on the investment. (Bell, 2003) uses a reverse auction protocol to determine where the replica should be created.

e.g. following rule is used in (Carman, 2002). A file request (FR) is considered to be an n-tuple of the form:

$FR_i = \langle t_i, o_i, g_i, n_i, r_i, s_i, p_i \rangle$

Where,

t_i: timestamp at which the file was requested

o_i, g_i and n_i: together represents the logical file being requested (o_i is the Virtual Organisation to which the file belongs, g_i is the group and n_i is the file identification number)

r_i and s_i: represents the element requesting and supplying the file respectively.

p_i: represents the price paid for the file (price could be virtual money)

To maximise the profit the future value of the file is defined over the average life time of the file storage T_{av}.

$$V(F, T_k) = \sum_{i=k+1}^{k+n} p_i \partial(F, F_i) \partial(s, s_i)$$

Where, V represents the value of the file. p_i represents the price paid for the file. s is the local storage element and F represents the triple (o, g, n). ∂ is a function that returns 1 if the arguments are equal and 0 if they differ. The investment cost is determined by the difference in cost between the price paid and the expected price if the file is sold immediately.

As the storage space of the site is limited, it must make a choice before replicating a file that whether it is worth to delete an existing file. Thus, the investment decision between purchasing a new file and keeping an old file depends on the change in profit between the two strategies.

Cost Estimation based

Cost estimation model (Lamehamedi, 2003) is very similar to the economic model. The cost-estimation model is driven by the estimation of the data access gains and the maintenance cost of the replica. While the investment measured in Economic models (Carman, 2002) (Bell, 2003) are only based on data access, it is more elaborate in the cost estimation model. The cost calculations are based on network latency, bandwidth, replica size, runtime accumulated read/write statistics (Lamehamedi, 2003) etc.

Replication Strategy for Update request

Synchronous

In synchronous model a replica is modified locally. The replica propagation protocol then synchronizes all other replicas. However, it is possible that other nodes may work on their local replicas. If such a conflict occurs, the job must be redone with the latest replica. This is very similar to the synchronous approach discussed in the distributed DBMS section.

Asynchronous

Various consistency levels are proposed for asynchronous replication. Asynchronous replication approaches are discussed as follows (Dullmann, 2001):

(i) Possible inconsistent copy (Consistency level: -1): The content of the file is not consistent to two different users. E.g. one user is updating the file while the other is copying it, a typical case of "dirty read problem".

(ii) Consistent file copy (Consistency level: 0): At this consistency level, the data within a given file corresponds to a snapshot of the original file at some point in time.

(iii) Consistent transactional copy (Consistency level: 1): A replica can be used by clients without internal consistency problems. But, if the job needs to access more than one file, then the job may have inconsistent view.

The following figure shows the classification of replication scheme discussed above. The major classification criterion is the update characteristics of transaction.

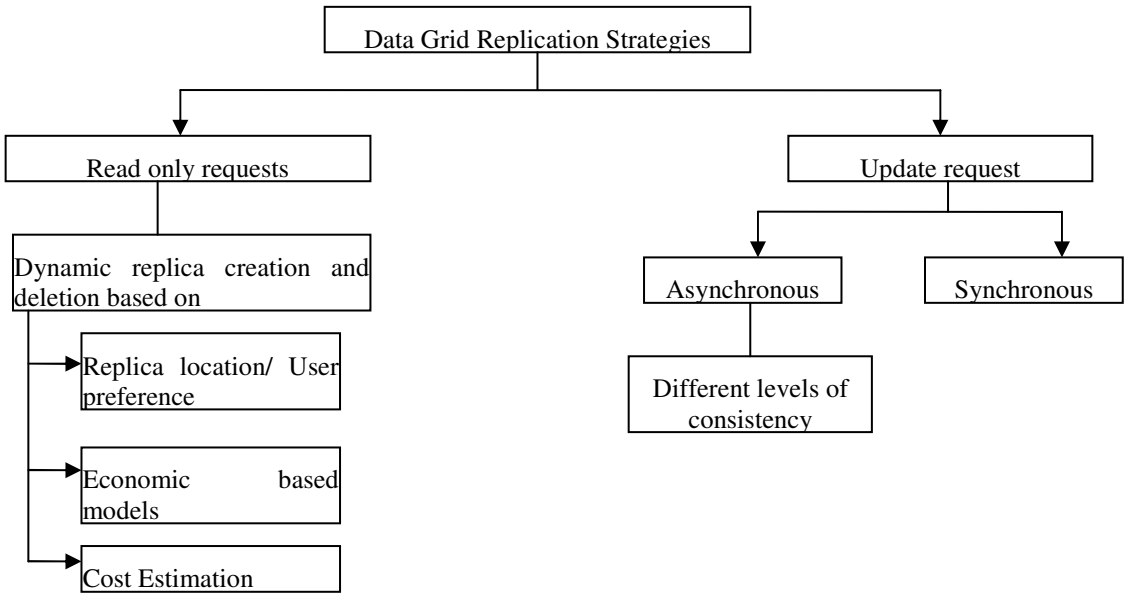

Figure 7: Classification of replication scheme in Data Grids.

Data Grid Replication Schemes

Overview of Replication studies in Data Grids are as follows along with brief explanation of each strategy:

Vazhkudai et. al. (2001) proposes a Replica selection scheme in Globus Data Grid. The method optimises the selection of replica in the dynamic Grid environment. A High level replica selection service is proposed. Information such as replica location and user preferences are considered to select the suitable replica from multiple replicas.

Lamehamedi et. al. (2003) proposes a method for dynamically creating replicas based on cost-estimation model. Replication decision is based on gains of creating a replica against creation and maintenance cost of the replica.

In economy-based replica protocols, Carman et. al. (2002) aims to achieve global optimisation through local optimisation with the help of emerging market place behaviour. The paper proposes a technique to maximise the profit and minimise the cost of data resource management. The value of the file is defined as the sum of the future payments that will be received by the site.

Another economy-based approach for file replication proposed by Bell et. al. (2003) dynamically creates and delete replica of files. The model is based on reverse Vickery auction where the cheapest bid from participating replica sites is accepted to replicate the file. It is similar to the work in (Carman, 2002) with the difference in predicting the cost and benefits.

Consistency issues have received limited attention in Data Grids. Dullmann et. al. (2001) proposes a Grid Consistency Service (GCS). GCS uses Data Grid services and supports replica update synchronisation and consistency maintenance. Different levels of consistency is proposed, starting from level -1 to level 3 in increasing order of strictness.

Lin et. al. (2005) proposes various policies for selecting a server for data transfer. Least Cost Policy chooses the server with minimum cost from the server list. Minimize Cost and Delay Policy considers delay in transferring the file in addition to the cost of transferring. A 'scoring function' is calculated from time and delay in replicating files. The file is replicated at the site with highest score. Minimize Cost and Delay with Service Migration policy considers the variation in service quality. If the site is incapable of maintaining the promised service quality, the request can be migrated to other sites.

WORLD WIDE WEB (WWW)

WWW has become a ubiquitous media for content sharing and distribution. Applications using the Web spans from small business applications to large scientific calculations. Download delay is one of the major factors that affect the client base of the application. Hence, reducing latency is one of the major research focus in WWW. Caching and replication are two major techniques used in WWW to reduce request latencies. Caching is typically on the client side to reduce the access latency, whereas replication is implemented on the server side so that the request can access the data located in a server close to the request. Caching targets reducing download delays and replication improves end-to-end responsiveness. Every caching technique has an equivalent in replica systems, nut the vice versa is not true.

Large volumes of request at popular sites may be required to serve thousands of queries per second. Hence, web servers are replicated at different geographical locations to serve request for services in a timely manner. From the users' perspective, these replicated web servers act as a single powerful server. Initially servers were manually *mirrored* at different locations. But continuously increasing demand of hosts has motivated the research of dynamic replication

strategy in WWW. Following major challenges can be easily identified in replicated systems in the Internet (Loukopoulos, 2002):

(i) How to assign a request to a server based on a performance criterion

(ii) Number of placement of the replica

(iii) Consistency issues in presence of update requests

Here we would briefly like to mention about Akamai Technologies (http://www.akamai.com). Akamai technologies have more than 16000 servers located across the globe. When a user requests a page from the web server, it sends some text with additional information for getting pages from one of the Akamai servers. The users' browser then requests the page from Akamai's server, which delivers the page to the user.

Most of the replication strategy in Internet uses a Primary-copy approach (Baentsch, 1996, http://www.newcastle.research.ec.org/cabernet/workshops/plenary/3rd-plenary-papers/13-baentsch.html and Khan, 2004). Replication techniques in (Baentsch, 1996 and http://www.newcastle.research.ec.org/cabernet/workshops/plenary/3rd-plenary-papers/13-baentsch.html) uses a Primary Server (PS) and Replicated Servers (RS). In (http://www.newcastle.research.ec.org/cabernet/workshops/plenary/3rd-plenary-papers/13-baentsch.html) the main focus is on maintaining up-to-date copies of documents in the WWW. A PS enables the distribution of most often request documents by forwarding the updates to the RS as soon as the pages are modified. A RS can act as replica server for more than one PS. RS can also act a cache for non replicated data. RS also reduce the load on the web servers, as they can successfully answer requests.

Replica management in Internet is not as widely studied and understood as in other distributed environment. We believe that due to changed architectural challenges in the Internet, it needs special attention. Good replication placement and management algorithms can greatly reduce the access latency.

DISCUSSION AND ANALYSIS

In this section we discuss different data storage technologies such as Distributed DBMS, P2P systems and Data Grids for different data management attributes.

Data Control: In distributed DBMS, the data is owned mostly by a single organisation, and hence can be maintained with central management policies. In P2P systems, control of the data is distributed across sites. Site where the data is stored is thought as owning the data and there is no obligation to follow a central policy for data control. Considering the most widely used Data Grid environment (LHC experiment), the data is produced at a central location but is hierarchically distributed to processing sites.

Autonomy: Distributed DBMSs are usually tightly coupled, mainly because they belong to single organisation. Hence the design choices depend on one another, and the complete system is tightly integrated and coupled. P2P systems are autonomous, as there is no dependency among any distributed sites. Each site is designed according to independent design choices and evolves without any interference of each other. In Data Grids, sites are autonomous of each other, but the typical characteristic is that they mostly operate in trusted environment.

Table 5: Comparison of different storage and content management systems.

Systems / Attributes	Distributed DBMS	P2P Systems	Data Grid	WWW
Data control	Mostly central	Distributed	Hierarchical	Mostly central
Autonomy among sites	Tightly coupled	Autonomous	Autonomous, but in trusted environment	Tightly coupled
Load distribution	Central and easy	Decentralised	Hierarchical	Central
Update performance	Well understood and can be controlled	Difficult to monitor	Not well studied yet (most studies are in read-only environment)	Mostly read content
Reliability	Can be considered during designing and has a direct relation with performance (in replication scenario)	Difficult to account for during system design. (as a peer can anytime disconnect from the system)	Intermediate	Central management. Hence can be considered at design time
Heterogeneity	Mostly homogeneous environment	Heterogeneous environment	Intermediate, as the environment is mostly trusted	Mostly homogeneous
Status of replication strategies	Read and update scenarios are almost equivalent	Mostly read environment	Mostly read but does need to update depending on the application requirement.	Mostly read environment. Lazy replication

Load Distribution: Load distribution directly depends on the data control attribute. If the data is centrally managed it is easy to manage the load distribution among distributed servers, as compared to distributed management. It is easy to manage the distributed data in DBMS environment as compared to P2P systems, because central policies can be implemented in DBMSs for data management, while it is virtually impossible to implement a central management policy in P2P systems.

Update Performance: Update performance in databases is easy to monitor and analyse during the database design (again, due to the fact that it is centrally designed). Databases, in general, have well defined data access interface and access pattern. Due to decentralised management and asynchronous behaviour of P2P systems, it may be difficult to monitor update performance

in such systems. In Data Grids, applications are mainly driven by read-only queries, and hence update performance is not well studied and understood. But, with advancement in technology, applications will need to update stored data in Data Grids as well. Hence, there is a need to study update performance in greater detail.

Reliability: As distributed DBMS work under a central policy, the downtime of a particular site can be scheduled and load of that site can be delegated to other sites. Thus DBMS systems can be designed for a guaranteed Quality of Service (QoS). P2P systems' architecture is dynamic. Sites participating in P2P systems can join and leave the network according to their convenience and hence cannot be scheduled. In Grids, though the architecture is dynamic, research has focussed on providing a QoS guarantee and some degree of commitment towards the common good.

Heterogenity: Distributed DBMSs typically work in homogeneous environment, as it is built bottom-up by the designer. P2P systems can be highly heterogenous in nature, since sites are autonomous and are managed independently. As shown in Figure 6, Data Grids have hierarchical architecture and individual organisations and institutes may choose for homogeneous environment, but different participants may opt for heterogenous component or policies.

Replication strategies: Database designers pay attention for update requests as well as performance of read-only queries. At the same time applications also demand update transactions and read-only queries almost equally. P2P systems are designed for applications requiring only file sharing. Thus, P2P systems mostly focus on read-only queries. Data Grids, so far, have mainly focussed on read-only queries, but the importance of write queries is also being realised and is attracting research interest.

As we are discussing data management systems, we would briefly want to mention the preservation work done by Stanford Peers Group (http://www-db.stanford.edu/peers/) for the sake of completeness. Data preservation mainly focuses on archiving the data for long term, e.g. in digital libraries. Data replication in such an environment can improve the reliability of the data. Such systems should be able to sustain long term failures. Replication can help in preserving online journal archive, white papers, manuals etc. against single system failure, natural disaster, theft etc. Data trading is one such technique proposed in (Cooper, 2002) to increase the reliability of preservation systems in P2P environment.

The purpose of this paper is to gather and present the replication strategies present in different architectural domains. This paper will help researchers working in different distributed data domains to identify and analyse replication theories present in other distributed environments and borrow some of the existing theories, which best suits them. Replication theories have been studied and developed for many years in different domains, but there have been lack of a comparative study. In this paper we had presented the state-of-the-art and research direction of replication strategies in different distributed architectural domains, which can be used by researchers working in different architectural areas.

SUMMARY AND CONCLUSION

We presented different replication strategies in distributed storage and content management systems. With changing architectural requirements, replication protocols have also changed and

evolved. A replication strategy suitable for a certain application or architecture may not be suitable for other. The most important difference in replication protocols is due to consistency requirements. If an application requires strict consistency and has lots of update transactions, replication may reduce the performance due to synchronisation requirements. But, if the application requires read-only queries, the replication protocol need not worry for synchronisation and performance can be increased. We would like to conclude by mentioning that though there are continuously evolving architectures, replication is now a widely studied area and new architectures can use the lessons learned by researchers in other architectural domains.

REFERENCES

Baentsch M., Molter G. and Sturm P. (1996). Introducing Application-level Replication and Naming into today's Web, *International Journal of Computer Networks and ISDN Systems*, *28*(7), 921-930.

Bell W. H., Cameron D. G., Carvajal-Schiaffino R., Millar A. P., Stockinger K., Zini F. (2003). Evaluation of an Economy-Based File Replication Strategy for a Data Grid, *Proccedings of 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, Tokyo Japan, May 2003, IEEE Computer Society Press.

Bernstein P. A., Hadzilacos V. and Goodman N. (1987). *Concurrency Control and Recovery in Database Systems*, Massachusetts, Addision-Wesley Publishers.

Bhagwan R., Moore D., Savage S., and Voelker G. M. (2002). Replication Strategies for Highly Available Peer-to-Peer Storage, *Proceedings of International Workshop on Future Directions in Distributed Computing*, Italy, June 2002.

Birman K. P. and Joseph T. A. (1987). Reliable communication in the presence of failures, *ACM Transactions on Computer Systems*, *5*(1), 47-76.

Carman M., Zini F., Serafini L. and Stockinger K. (2002). Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid. *Proceedings of the 1st IEEE/ACM International Conference on Cluster Computing and the Grid (CCGrid)*, (pp: 340-345), Berlin, Germany, May 2002, IEEE Computer Society Press.

Cohen E., Shenker S. (2002). Replication Strategies in Unstructured Peer-to-Peer Networks, *Proceeding of Special Interest Group on Data Communications (SIGCOMM),* (pp: 177-190), Pittsburg, USA, August 2002.

Cooper B. and Garcia-Molina H. (2002). Peer-to-peer data trading to preserve information, ACM Transactions on Information Systems (TOIS), *20*(2), April 2002, 133-170.

Domenici A., Donno F., Pucciani G., Stockinger H., Stokinger K. (2004). Replica Consistency in a Data Grid, *Nuclear Instruments and Methods in Physics Research; Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, *534*(1-2), November 2004, 24-28, Elsevier publisher.

Dullmann D., Hosckek W., Jaen-Martinez J., Segal B., Samar A., Stockinger H. and Stockinger K.(2001). Models for Replica Synchronisation and Consistency in a Data Grid, *Proceeding of 10th IEEE International Symposium on High Performance and Distributed Computing (HPDC)*,

(pp: 67-75), San Francisco, August 2001.

Foster I.and Kesselman C. (Ed.). (2004). *The Grid: Blueprint for a New Computing Infrastructure*, Second Edition, Morgan Kaufmann Publishers, USA.

Gray J., Helland P., O'Neil P., and Shasha D. (1996). The Dangers of Replication and a Solution. *Proceedings of International Conference on Management of Data (ACM SIGMOD)*, (pp: 173-182), Montreal, Canada, June 1996.

Helal A. A., Hedaya A. A. and Bhargava B. B. (1996). *Replication Techniques in Distributed Systems*, Boston, Kluwer Academic Publishers.

Khan S. U. and Ahmad I. (2004). *Internet Content Replication: A Solution from Game Theory* (Technical Report CSE-2004-5), Department of Computer Science and Engineering, University of Texas at Arlington.

Kistler J. J. and Satyanarayanan M. (1992). Disconnected Operation in the Coda File System, *ACM Transactions on Computer Systems*, *10*(*1*), 3-25.

Lamehamedi H., Shentu Z., Szymanski B.and Deelman E. (2003). Simulation of Dynamic Data Replication Strategies in Data Grids, *Proceedings of the 17th International Parallel and Distributed Processing Symposium* (PDPS), Nice, France, IEEE Computer Society.

Lin H. and Buyya R. (2005). *Economy-based Data Replication Broker Policies in Data Grids*, Unpublished BSc (Honours) thesis, Department of Computer Systems and Software Engineering, The University of Melbourne.

Liskov B., Ghemawat S., Gruber R., Johnson P., Shrira L.and Williams M. (1991). Replication in the Harp File System, *Proceedings of 13th ACM Symposium on Operating Systems Principles* (pp: 226-238), ACM Press.

Loukopoulos T., Ahmad I. and Papadias D. (2002), An Overview of Data Replication on the Internet, *Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks* (ISPAN) (pp: 694-711), IEEE Computer Society.

Lv Q., Cao P., Cohen E., Li K., Shenker S. (2002). Search and Replication in Unstructured Peer-to-Peer Networks, *Proceeding of the 16th Annual ACM International conference on Supercomputing* (pp: 84-95) NY, USA, June 2002.

Oram A. (Ed.). (2001). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, ISBN: 0-596-00110-X, Sebastopol, USA, O'Reilly publishers.

Paris J.-F. (1986). Voting with Witnesses: A Consistency Scheme for Replicated Files, *Proceedings of Sixth International Conference on Distributed Computing Systems* (pp: 606-612), Washington DC, USA, IEEE Computer Society.

Parrington G. D., Shrivastava S. K., Wheater S. M. and Little M. C. (1995). The Design and Implementation of Arjuna, *USENIX Computing Systems Journal, 8*(2), 255-308.

Pless, V.(1998). *Introduction to the theory of error-correcting codes*. 3rd edition, New York, USA, John Wiley and Sons.

Raman P., Livny M.and Solomon M. (1998). Matchmaking: Distributed Resource Management for High Throughput Computing, *In Proceedings of 7th IEEE Symposium on High Performance Distributed Computing (HPDC)* (pp: 140-146), July 1998, IEEE Computer Society Press.

Ranganathan K. and Foster I. (2001). Identifying dynamic Replication Strategies for a High-Performance Data Grid, *Proceedings of International Workshop on Grid Computing* (pp: 75-86), Denver, USA, November 2001.

Sidell J., Aoki P. M., Barr S., Sah A., Staelin C., Stonebraker M.and Yu A. (1996). Data Replication in Mariposa (pp: 485-494), *Proceedings of 17th International Conference on Data Engineering*, New Orleans, USA, February 1996.

Siegel A., Birman K., and Marzullo K. (1990). Deceit: A Flexible Distributed File System (Technical Report No. 89-1042), Department of Computer Science, Cornell Univ., November 1989 (also in *Proceedings of USENIX Conference*, Anaheim USA, pp. 51-62).

Sybase FAQ, http://www.faqs.org/faqs/databases/sybase-faq/part3/

Triantafillou P. and Neilson C. (1997). Achieving Strong Consistency in a Distributed File System, *Proceedings of IEEE Transactions on Software Engineering*, *23*(1), 35-55.

Vazhkudai S., Tuecke S., Foster I. (2001). Replica Selection in the Globus Data Grid, *Proceedings of the International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001)*, (pp: 106-113), May 2001, IEEE Computer Society Press.

Venugopal S., Buyya R., and Ramamohanarao K. (2005). A Taxonomy of Data Grids for Distributed Data Sharing, Management and Processing (Technical Report, GRIDS-TR-2005-3), Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia.

URLS:

Ahmed R., DeSmedt P., Du W., Kent W., Ketabchi M., Litwin W., Rafii A. and Shan M., "Using an Object Model in Pegasus to Integrate Heterogeneous Data" http://www.bkent.net/Doc/usobpeg.htm

Baentsch M., Baum L., Molter G., Rothkugel S. and Sturm P., "Caching and Replication in the World-Wide Web", available from http://www.newcastle.research.ec.org/cabernet/workshops/plenary/3rd-plenary-papers/13-baentsch.html

Baumgartel P., Oracle Replication: An Introduction, http://www.nyoug.org/200212baumgartel.pdf
http://www.akamai.com

http://www.dbmsmag.com/9705d15.html

http://www.dothill.com/tutorial/tutorial.swf

http://www-db.stanford.edu/peers/