Proceedings of
National Conference on Innovative Trends in Computer Science Engineering (ITCSE-2015)
held at BRCMCET, Bahal on 4th April 2015

Replication in Distributed System Network

[1]Santosh Kumar Srivastava , [2]Preeti Poonia , [3]Seema
[1]Sr. System Administrator, [2,3]M.Tech Scholar
[1,2,3]BRCM College of Engineering & Technology,Bahal, Bhiwani

***Abstract* – In Distributed System several nodes are connected with each other and they co-operate their execution of instructions to get a single goal. During this execution they may be in a need of common set of data. As instructions are executing at different machines, to ensure the availability of data, replication is required. Replication increases availability and finally concurrency increases. Though replication supports concurrency, to maintain the consistency of data extra cost has to be paid. Operations being performed on data may be in read-only mode or read and write mode. The conflict mode operations are handled carefully so that consistency of data may be preserved. Replication may be done on different physical machines or on a single machine. If replication is taking place replica managers are required to maintain the consistency of data. Replica Managers are connected with each other. Clients interact with the replica managers to get a copy of desired data. This paper is shedding light on the various aspects of replication strategies in Distributed System.**

Keywords – Distributed System, Replication, Consistency Availability, Replica Manager.

I. INTRODUCTION

In Distributed System several machines are connected with each other and they coordinate their execution to achieve a dedicated task. Machines in this computing may suffer from heterogeneities in terms of both hardware and software. Transaction in distributed system may be flat or nested. Flat transaction is one that starts and terminates on a single machine. Nested transaction is one that initiates on a machine and creates sub transactions. Distributed transaction is one that takes place on different machines. A distributed transaction may need a common set of data on each machine. This requirement can be fulfilled into two ways: (1) Providing a copy of common data on each machine or (2) there is a machine containing several copies of the common data. Let every transaction is locking the data in read only mode then concurrency being provided by the replication does not require any additional cost. If transactions are locking the data in conflict mode then consistency and concurrency requires additional cost

	read	**write**
read(Q	OK	NO
write(	NO	NO

Fig.1. *Compatibility Matrix*

In conventional database, compatibility matrix is given as above.

read(Q): A transaction wants to read the value of data Q.
write(Q): A transaction wants to write the value of data Q.

If two transactions issue read(Q) simultaneously, lock to Q will be granted else not. In conflict mode operation, a schedule is required that defines order of transactions on data Q.

Replicas of a data cannot be in different states at a time. To maintain consistency communication among the replicas must take place in spite of whether they are residing: on different machines or on same machine.

I. REPLICATION SCENARIO IN DISTRIBUTED SYSTEM

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 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:

- Read-only queries
- Update transactions
- 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) [6, 7]. 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 given 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.

II. Advantage Of Replication

Service enhancement is the motivational factor behind replication. Replication encourages the followings:

- **Performance Enhancement**
- **Increased Availabilty**
- **Fault Tolerant**

Let us consider that we are maintaining *n* replicas of a data object and *p* defines its probability of unreachability. Then total availability is stated as $(1-p^n)$. Further replication supports the following cases:

- Server Failure
- Network Partitioning
- Disconnected Operation

Highly available data is not necessarily strictly correct data. It may be out of date. Two transactions may perform operations in conflicting modes and can leave the data in inconsistent state. A fault tolerance mechanism should be there to provide an up to date copy of the data.

If replication is made on *f* servers and (*f-1*) servers crash then we will have one server to provide us the desired data. Services based on replication should offer location transparency. Location transparency says that client should not normally have to be aware that multiple copies of data exist.

Next section of this paper is describing the challenges associated with data replication.

III. Challenges In Data Replication

A. Data Consistency

Maintaining data integrity and consistency in a replicated environment is of prime importance. High precision applications may require strict consistency.

B. 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 requests due to consistency requirements.

C. 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.

D. 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.

IV. Replication Technique In Distributed System

There are number of techniques for file replication that are used to maintain data consistency. Replication services maintain all the copies or replicas having the same versions of updates. This is known as maintaining consistency or synchronization [3]. Replication techniques to provide consistency can be divided into two main classes: [10]

• Optimistic- These schemes assume faults are rare and implement recovery schemes to deal with inconsistency.

• Pessimistic- These schemes assume faults are more common, and attempt to ensure consistency of every access.

Schemes that allow access when all copies are not available use voting protocols to decide if enough copies are available to proceed.

A. Pessimistic Replication

This is a more conservative type scheme using prime site techniques, locking or voting for consistent data update. As this approach assumes that failure is more common it guards against all concurrent updates. An update cannot be written if a lock cannot be obtained or if majority of other sites cannot be queried. In doing so, you will sacrifice data availability. The pessimistic model is a bad choice where frequent disconnections network and network partitions are common occurrence [3].

B. Optimistic Replication

This approach assumes that concurrent updates or conflicts are rare [3]. This scheme allows concurrent update, updates can be done at any replica or copy. This increases the data availability. However, when conflicts do occur, special action must be taken to resolve the conflict and merge the concurrent updates into a single data object. The merging is referred to as conflict resolution. When conflicts do occur, many of them can be resolved transparently and automatically without user involvement [3]. This approach is used for mobile computing.

V. Replication Reconciliation

Updates and modifications must be propagated to all replicas. This can be done immediately when the update occurs or it can be done at a scheduled interval later. Immediate propagation to all the replicas is fast but it is expensive to do so if it is not important. Alternatively updates can be done later, more like a batch processing. This is a periodic reconciliation, which allows propagation to occur when it is cheap or convenient. In systems which have disconnected operations, periodic reconciliation must be supported, as the immediate reconciliation will fail when the systems is disconnected [1, 2, 4, 5].

VI. Replication Models

There are three basic replication models the master-slave, client-server and peer-to-peer models.

A. *Master-slave model*

In this model one of the copy is the master replica and all the other copies are slaves. The slaves should always be identical to the master. In this model the functionality of the slaves are very limited, thus the configuration is very simple. The slaves essentially are read-only. Most of the master-slaves services ignore all the updates or modifications performed at the slave, and "undo" the update during synchronization, making the slave identical to the master [3]. The modifications or the updates can be reliably performed at the master and the slaves must synchronize directly with the master.

B. *Client-server model*

The client-server model like the master-slave designates one server, which serves multiple clients. The functionality of the clients in this model is more complex than that of the slave in the master-slave model. It allows multiple inter-communicating servers, all types of data modifications and updates can be generated at the client. One of the replication systems in which this model is successfully implemented is

Coda. Coda is a distributed file system with its origin in AFS2. It has many features that are very desirable for network file systems [8]. Optimistic replication can use a client-server model. In Client- server replication all the updates must be propagated first to the server, which then updates all the other clients. In the client-server model, one replica of the data is designated as the special server replica. All updates created at other replicas must be registered with the server before they can be propagated further. This approach simplifies replication system and limits cost, but partially imposes a bottleneck at the server [11, 12]. Since all updates must go through the server, the server acts as a physical synchronization point [11]. In this model the conflicts which occur are always be detected only at the server and only the server needs to handle them. However, if the single server machine fails or is unavailable, no updates can be propagated to other replicas. This leads to inconsistency as individual machines can accept their local updates, but they cannot learn of the updates applied at other machines. In a mobile environment where connectivity is limited and changing, the server may be difficult or impossible to contact, while other client replicas are simple and cheap to contact. The peer model of optimistic replication can work better in these conditions [11].

C. *Peer-to-peer model*

The Peer-to-peer model is very different from both the master-slave and the client-server models. Here all the replicas or the copies are of equal importance or they are all peers. In this model any replica can synchronize with any other replica, and any file system modification or update can be applied at any replica. Optimistic replication can use a peer-to-peer model. Peer-to-peer systems allow any replica to propagate updates to any other replicas [8, 9, 10]. The peer-to-peer model has been implemented in Locus, Rumor and in other distributed environments such as xFS in the NOW project. Peer-to-peer systems can propagate updates faster by making use of any available connectivity. They provide a very rich and robust communication framework. But they are more complex in implementation and in the states they can achieve [11]. One more problem with this model is scalability. Peer models are implemented by storing all necessary replication knowledge at every site thus each replica has full knowledge about everyone else. As synchronization and communication is allowed between any replicas, these results in exceedingly large replicated data structures and clearly does not scale well. Additionally, distributed algorithms that determine global state must, by definition, communicate with or hear about (via gossiping) each replica at least once and often twice. Since all replicas are peers, any single machine could potentially affect the outcome of such distributed algorithms; therefore each must participate before the algorithm can complete, again leading to potential scaling problems [3]. Simulation studies in the file system arena have demonstrated that the peer model increases the speed of update propagation among a set of replicas, decreasing the frequency of using an outdated version of the data [5, 6].

VII. **Architectural Model For Replica Management**

The system model assumes that no network partitioning occurs in system. Whenever a client needs a replica, clients sends request to Front End (FE). On behalf of client, the FE forwards this request to Replica Manager (RM). Replica Managers (RMs) communicates each other to ensure availability of an up to date copy of replica.

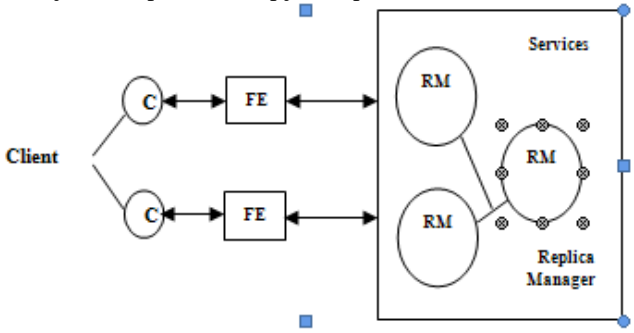

Fig.2. *Architectural Model for Replica Management*

VIII. NUMBERS OF REPLICA MANAGER

In a system, number of RMs may be constant or dynamic. Multiple clients may be connected with a single FE. If there exists multiple requests for replica at both the FE and the RM may be overloaded. This additional load on RMs may decrease the efficiency of entire system if constant RMs exists in the system. The RMs can be dynamically created there. In dynamic environment we have a better possibility of QoS. Further in a system, mapping of client with FE may be one to one or one to many (one client to many FEs).

IX. CACHING AT FRONT END

Front end may cache a replica. This caching scheme is beneficial when a client requests for replica and at FE there exist up to date copy of replica then FE sends back this copy to the replica instead of forwarding this request to RM. Replica may be temporal or non-temporal. Cached replica has the life time and if it is expired then FE may delete this from its cache and requests to RM to provide an up to date copy of replica.

X. ROLE OF REPLICA MANAGER

A. *Coordination*

Replica Manager communicates each other to process a request. This occurs in two different cases: when requested RM is unable to process the request of FE due to heavy load, it transfers the request to another RM. Second case is when RMs has to reach on a final value of the replica. Both FE and RM maintain ordering of requests at this level:

- Ordering at Front End
  1). *FIFO ordering:* If client sends request r_1 at t_1 time and r_2 at t_2 time then a correct FE will process the r_1 first if $t_1 < t_2$.
  2). *Casual ordering*
  If a client is sending a request r_1 at t_1 time and r_2 at t_2 time then a correct FE will process r_1 before r_2 if r_1->r2. The '->' indicates happening before relationship.
  3). *Total ordering*
  If a client is connected with multiple FEs and an FE is processing r_1 before r_2 then all other FEs with whom client is connected performs same ordering.

- Ordering at Replica Manager:

1). FIFO ordering

The RM executes the request in same manner as FE sends. If FE forwards a request r_1 at t_1 time and r_2 at t_2 time to an RM and $t_1<t_2$ then RM process r_1 before r_2.

2). Casual ordering

If an FE serves two request r_1 and r_2 as: $r_1 \rightarrow r_2$ i.e. r_1 is happening before r_2 then correct RM will process r_1 before r_2.

3). Total ordering

If an FE is connected with more than one RM and an RM is executing r_1 before r_2 then all the other RMs will be executing r_1 before r_2.

B. Execution

Updation submitted by client at RM is executed tentatively so that they can undo its effect later.

C. Agreement

There exists an agreement in between RMs that whether transaction committed by client will be aborted or committed.

D. Response

If an FE is connected with multiple RMs then for a request, FE receives more than one response. Now it is up to the FE which response will be used. One possible parameter will be time. Similarly a client may be connected with multiple FEs. In this case a client will receive more than one response for a request. When multiple responses are received by a client, client decides which one should be used or not. This selection may be dependent on time. Model represented in fig (1) is a generalization. Some variations on this model are possible. These variations are known as passive (primary backup) replication model and active replication model. Next two sections are shedding light on these two variations.

XI. Passive Replication Model

In the passive or primary backup model of replication, there is a single primary replica manager and one or more secondary replica managers- called backups or slaves. In its pure form, the FE communicates only with the replica manager (primary) to obtain services. The primary replica manager executes the operation and sends copies to the backups (updated copies). Following are the steps invloved in it.

A. **Request**

The FE issues the request containing the unique identifier, to the primary replica manager.

B. **Coordination**

The primary takes each request automatically in the order in which they are received. It checks the unique identifier; in case it has already executed the request and if so it simply re-sends the response.

C. **Agreement**

If the request is an update then the primary sends the updated state, the response and the unique identifier to all the backups. The backup sends an acknowledgment to the primary.

D. **Response**

The primary responds to the front end and the FE sends the response back to clients.

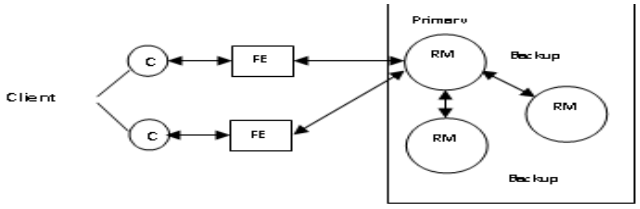

Fig.3. *Passive Replication Model*

In case of any failure in primary replication one of the secondary becomes the primary. Passive backup model is being presented in figure 4[10].

XII. Active Replication

Front ends multicast their request to group of replica managers. The RMs process the request (independent but identically) and reply to the FE.

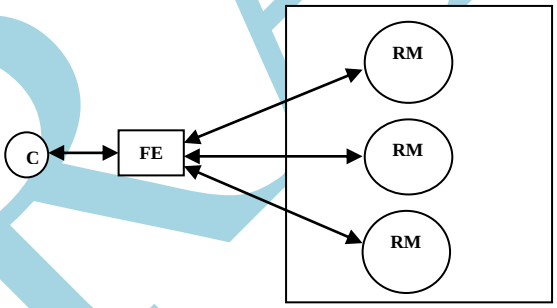

Fig.4. *Passive Replication Model*

If any replica manager caches then it has no impact on overall performance of the services, since the remaining replica managers continue to respond in normal way. Under active replication, the sequences of events are as follows:

A. Request

The FE attaches the unique identifier to the request and multicast it to the group of replica managers using totally ordered, reliable multicast primitive. The FE is assumed to fail by crashing at worst. It does not issue the next request until it has received the response.

B. Coordination

The group communication system delivers the request to the correct replica manager in the total order.

C. Execution

Every replica manager executes the request. Since they are state machine and since request are delivered in the same total order, correct replica managers process all the requests identically. The response contains the clients unique request identifier.

D. Agreement

No agreement phase is needed, because of multicast delivery semantics.

E. Response Each replica manager sends its response to the FE. The number of replies that the front end collects depend upon the failure assumption on the multicast algorithm[10].

XIII. CONCLUSION

Replication increases availability and finally concurrency increases. To ensure consistency among existing replicas is cost consuming. This paper discusses various aspect of replication in Distributed System. We have many examples where replication is playing vital role. A tradeoff exists among the degree of concurrency and the cost consumed in maintaining the consistency of replicas. If RMs fail on reaching an agreement, the updation submitted by the clients are ignored. Maintaining replicas of non-temporal data is easier than maintaining replicas of temporal data. Further read-only operations are more supportive to replication than write operation.

REFERENCES

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

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

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

[4]. D. Kempe, J. Kleinberg, and A. Demers. , "Spatial gossip and resource location protocols". Proceeding of 33rd ACM Symp. on Theory of Computing, 2001.

[5]. M.-J. Lin and K. Marzullo, " Directional gossip: Gossip in a wide area network," In Proceeding of 3rd European Dependable Computing Conference on Dependable Computing, 1999.

[6]. K. Shen, " Structure management for scalable overlay service construction," In Proceeding 1st USENIX/ACM Symp. on Networked Systems Design and Implementation, 2004.

[7]. R. van Renesse, K. Birman, and W. Vogels. Astrolabe, " A robust and scalable technology for distributed system monitoring, management, and data mining," ACM Transactions on Computer Systems, 21(2), May 2003.

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

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

[10]. Couloris, Dollimore and Kindberg, Distributed Systems Concepts and Design, 4e,2005, Pearson Education.

[11]. Foster I.and Kesselman C.,The Grid: Blueprint for a New Computing Infrastructure,2e,2004, Morgan Kaufmann Publishers, USA.Helal A. A., Hedaya A. A. and Bhargava B. B. (1996). Replication Techniques in Distributed Systems, Boston, Kluwer Academic Publishers.