Journal of Theoretical and Applied Information Technology
30th April 2014. Vol. 62 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPTIMIZATION OF TRANSACTION PROCESSING IN CLOUD DATABASES THROUGH A DATACENTER SYSTEM DESIGN CRACS TO IMPROVE REPLICA CONSISTENCY, AVAILABILITY AND SCALABILITY

[1]R. ANANDHI AND [2]DR. K. CHITRA

[1]Registered Scholar, Dept. of Computer Applications, SCSVMV University, Kanchipuram, TN, India
[2]Asst. Professor, Department of Computer Science, Govt. Arts College, Melur, Madurai, TN, India
E-mail: [1]anandhi78@yahoo.com, [2]manikandan.chitra@gmail.com

ABSTRACT

It is obvious that tremendous achievements have been emerged in IT industries at various sectors. If we have a keen look, all the technologies are revolving around a single word "DATA". All techniques are trying to improve the read and write of data from and to the database. Usually read and write are referred by common word called "transactions". Cloud Computing is one of those technologies that involves execution of Database transactions. This paper provides a system model called CRACS which maintains atomicity, isolation, consistency and durability of transactions at NOSQL databases which usually try to deviate from above said properties of transactions.
Keywords: *ACID, BASE, Cloud Computing, Consistency, Datacenter, Transactions.*

1. INTRODUCTION

Cloud computing is a new paradigm in which dynamically scalable virtualized computing resources are provided as a service over the Internet. As resources are limited, it is very important that cloud providers efficiently provide their resources [1]. The trust model for efficient reconfiguration and allocation of computing resources satisfy various user requests; moreover it collects and analyzes reliability based on historical information of servers in a Cloud data center. Then it prepares the best available resources for each service request in advance, providing the best resources to users [2]. Cloud computing is a new computing paradigm composed of Grid computing and Utility computing concepts together. It provides dynamically scalable virtualized computing resources as a service over the Internet and users pay for as many resources as they have used [3]. Due to limitations in software technologies and network bandwidth in the past, Cloud computing could not guarantee service levels and scope that needed to be delivered over the Internet. Nowadays, Cloud computing can provide various levels of service and functions over the Internet, as software and network technologies develop [4].

2. TYPES OF CLOUDS

Cloud computing has various advantages are improved performance, lower IT infrastructure costs, unlimited storage capacity, less maintenance and improved compatibility. The types of clouds are: a) **Public Cloud** is made available to the general public or a large industry group b) **Private Cloud** is operated solely for a single organization c) **Community Cloud's** infrastructure is shared by several organizations d) **Hybrid Cloud** is a composition of two or more clouds (private, community, or public) [5].

3. SERVICES OF CLOUDS

Cloud Computing denotes the hiring of resources like servers, memory, storage areas etc. Cloud itself is a network of virtualized servers or virtual data centers that can deliver powerful applications, platforms, and infrastructures as services over the Internet (ie) **IaaS** (Infrastructure as a Service is a provision model in which an organization outsource the equipment used to support operations, including storage, hardware, servers and networking components), **Paas**

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

(Platform as a Service is a delivery of a computing platform over the web), **SaaS** (Software as a Service is one of the methodologies of Cloud Computing, which is based on a "one-to-many" model whereby an application is shared across multiple clients) [5]. Recently a new service has been extended by Cloud called as **DaaS** (Database as a Service). There are two common deployment models: users can run databases on the cloud independently, using a virtual machine image, or they can purchase access to a database service, maintained by a cloud database provider. Of the databases available on the cloud, some are SQL-based and some use a NoSQL data model [6].

4. TRANSACTIONS

A **transaction** [7] comprises a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. Transactions in a database environment have two main purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
2. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the program's outcomes are possibly erroneous.

A simple transaction is usually issued to the database system in a language like SQL wrapped in a transaction, using a pattern similar to the following:

1. Begin the transaction.
2. Execute a set of data manipulations and/or queries.
3. If no errors occur then commit the transaction and end it.
4. If errors occur then rollback the transaction and end it.

5. PROPERTIES OF TRANSACTIONS

Normally the transactions of a database have to obey certain properties popularly termed as **ACID**. The term **ACID** stands for **A**tomicity, **C**onsistency, **I**ntegrity and **D**urability [8]. **Atomicity** refers to the ability of the DBMS to guarantee that either all of the tasks of a transaction are performed or none of them are. Atomicity states that database modifications must follow an "all or nothing" rule. If some part of a transaction fails, then the entire transaction fails, and vice versa. **Consistency** ensures that the database remains in a consistent state, despite the transaction succeeding or failing and both before the start of the transaction and after the transaction is over. **Isolation** refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction and helps to implement concurrency of database. **Durability** states that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. That means when users are notified of success, the transactions will be persist, not be undone and survive from system failure.

The running theme is "scaling out instead of scaling up," driven by the economics of PC commoditization, where scale out means adding more cheap components and scale up means adding more power and complexity to a small number of expensive components [9]. NOSQL (Not Only SQL) databases were developed from the ground up to be distributed, scale out databases. They use a cluster of standard, physical or virtual servers to store data and support database operations. To scale, additional servers are joined to the cluster and the data and database operations are spread across the larger cluster. Since commodity servers are expected to fail from time-to-time, NoSQL databases are built to tolerate and recover from such failure making them highly resilient [10]. NOSQL transactions obey BASE properties [11] which deviate from ACID concept. **Basically available** could refer to the perceived availability of the data. If a single node fails, part of the data won't be available, but the entire data layer stays operational. **Soft state** leads to the concept of data needing a period refresh. Without a refresh, the data will expire or be deleted. **Eventual consistency** means that updates will eventually ripple through to all servers, given enough time. Mostly the database designed by and support NOSQL will show only BASE properties. Thus transactions with BASE attributes are not expected to have immediate consistency. This paper is going to deal with a new architecture of datacenter so that all replicas of database are going to have the same content at any time "t" (ie) they are all consistent among themselves. Since the values at database are to be consistent, many research proposals are there to improve those expected parameters. Modern distributed data stores offer a choice of consistency models [12]. Weak consistency models are fast and guarantee "always on" behavior but provide limited

Journal of Theoretical and Applied Information Technology
30th April 2014. Vol. 62 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

guarantees. Stronger consistency models are easier to reason about but are slower and potentially unavailable. The choice of a consistency model has wide-ranging implications for application writers, operations management, and end-users. Yet, in light of its performance benefits, weak consistency is often considered acceptable. Eventual consistency—perhaps the most commonly deployed weak consistency model—is particularly weak: in the absence of new writes to a data item, reads will eventually return the same value [13].

6. DATACENTER

Regarding the components of Cloud Environment, **Datacenter** is the most important one because it hosts all the servers which store the required data available 24x7 [Figure 1]. Data centers are comprised of both server and networking infrastructure. The server portion of the infrastructure is now far down the road of commoditization - high-end enterprise-class servers have been replaced by large numbers of low-cost servers. Innovation in distributed computing and systems management software has enabled the unreliability of individual servers to be masked by the aggregated reliability of the system as a whole.

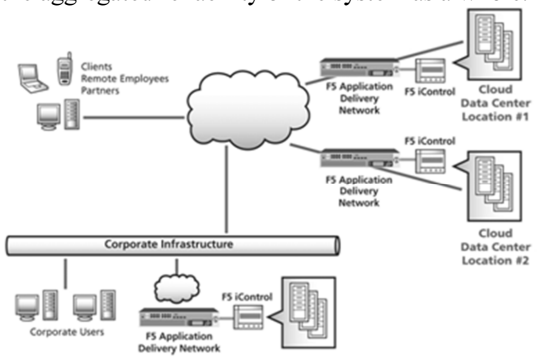

Figure 1: Datacenters in Cloud Environment

A data center is a facility used for housing a large amount of electronic equipment, typically computers and communications equipment. As the name implies, a data center is usually maintained by an organization for the purpose of handling the data necessary for its operations. For example, a bank may have a data center keeping all its customers' account information and transactions involving that data are carried out. Practically every company that is mid-sized or larger has some kind of data center with the larger companies often having dozens of data centers. Consistency Management of Replicas in Wireless Grid Environment consists in offering a service allowing controlling the management of the consistency in Wireless Grid [14]. A system design called Monsoon, a blueprint for commoditizing the networks of data centers used for "cloud" services where large numbers of servers cooperatively handle huge workloads [15].

7. DESCRIPTION OF CRACS

So our proposed system design **CRACS** (Figure 2) which stands for Cloud Replica with Availability, Consistency and Scalability tries to achieve ACID at NOSQL databases. Let us consider there are "n" servers available at a datacenter X. One of the servers is elected as a **Master Server**.

As the name implies, the Master Server is the Coordinator for all the transactions take place in that datacenter. The other "n-1" servers are designated as **Replica Servers** since they store the same replicas of various databases.

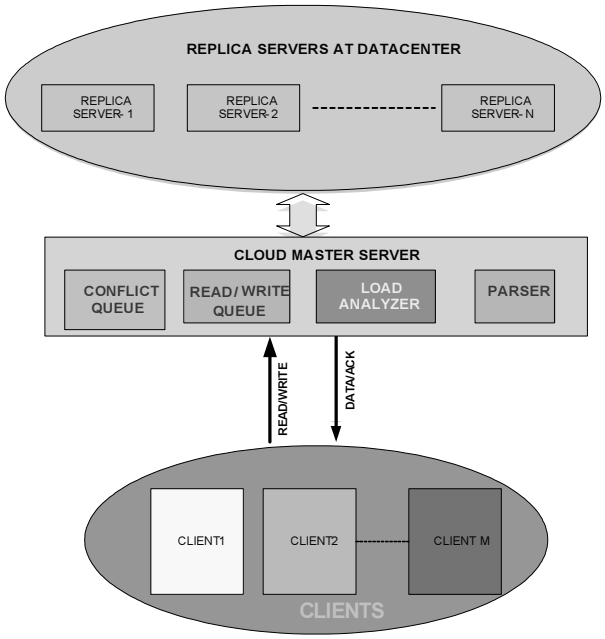

Figure 2: Cloud Replica with Availability, Consistency and Scalability (CRACS)

The Master Server is going to maintain the entire statistics about each replica server and doesn't have any other database storage. It is always watching the load at each replica server. Usually read transactions are more than the write transactions. But if we allow read first and then write, then the consistency can't be achieved. So CRACS will give priority to write than read. Here

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the Master Server is going to play roles like Parser, Queue Manager, Transaction Manager and Load Analyzer. Our database is build using Cassandra NOSQL database since it is massively scalable, partitioned row store, master less architecture, linear scale performance, no single points of failure, read/write support across multiple data centers & cloud availability zones. Initially the Replica Servers are arranged as a ring of servers so that each one knows its neighbor. But Master Server has to wait for the acknowledgement until it makes a round trip delay (ie) Master Server-Replica Server 1-Replica Server 2-----Replica Server n-Master Server. Since time is an important constraint, the ring topology has been withdrawn and CRACS is arranged as per the above Figure 2.

8. ALGORITHM

Let R_1, R_2, R_3…….R_n be the set of Replica Servers and T_1, T_2…..T_m be set of transaction available on the Queues. RQ means Read/Write Queue and CQ means Conflict Queue.

1. Repeat While (RQ is non-empty and CQ is non-Empty)

2. Read Transaction T_i.

3. If Transaction_Type(T_i) = ”Read” and Queue_Type(“RQ”) Then

    @Master Server find out the lightly loaded Replica Server say R_j from the statistics available.
    If @Replica Server R_j(lockbit=0) Then
        Set lockbit = 1;
        @ R_j Execute Transaction TXN(T_i);
        Message _Send(Client, Result);
        Set lockbit=0;
    Else
        Delete_Queue(RQ, T_i);
        Insert_Queue(CQ, T_i);
    End If;
    Goto Step-1;

4. Else If Transaction_Type(T_i) = ”Write” and Queue_Type(“RQ”) Then

    For k in 1 to Num_of_Replica_Servers Do
        Multicast(R_k, T_i);
    End For;
    If @Replica Server R_k(lockbit=0) Then
        lockbit=1;
        Message_Send(Master, ”Agree”);
        If (@Master Message_Count(“Agree”)) = Num_of_Replica_Servers Then
            @R_k Execute Transaction TXN(T_i);
            lockbit=0;
            Delete_Queue(RQ, T_i);
            Message _Send(Client, Result);
        Else
            Delete_Queue(RQ, T_i);
            Insert_Queue(CQ, T_i);
        End If;
    Else
        Delete_Queue(RQ, T_i);
        Insert_Queue(CQ, T_i);
    End If;
    Go to Step-1;
End If;

5. Else if Transaction_Type(T_i) = ”Read” and Queue_Type(“CQ”) Then

    @Master Server find out the lightly loaded Replica Server say R_j from the statistics available.
    If @Replica Server R_j(lockbit=0) Then
        Set lockbit = 1;
        @ R_j Execute Transaction TXN(T_i);
        Message _Send(Client, Result);
        Set lockbit=0;
    Else
        Delete_Queue(CQ, T_i);
        Message _Send(Client, “Failure”);
    End if;
    Go to Step 1;

6. Else If Transaction_Type(T_i) = ”Write” and Queue_Type(“CQ”) Then

    For k in 1 to Num_of_Replica_Servers Do
        Multicast(R_k, T_i);
    End For;
    If @Replica Server R_k(lockbit=0) Then
        lockbit=1;
        Message_Send(Master, ”Agree”);
        If (@Master Message_Count(“Agree”)) = Num_of_Replica_Servers Then
            @R_k Execute Transaction TXN(T_i);
            lockbit=0;
            Delete_Queue(RQ, T_i);
            Message _Send(Client, Result);
    Else

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Delete_Queue(CQ, T_i);
Message _Send(Client, Failure");
End if;

End if;
Go to Step 1;

End While;

9. WORKING OF CRACS

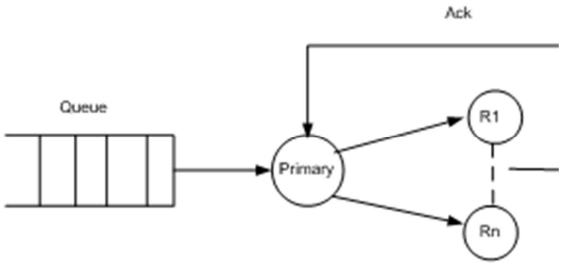

Figure 3: Write operation @ CRACS

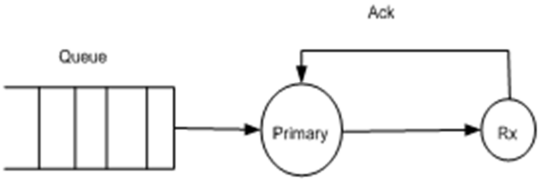

Figure 4: Read operation @ CRACS

The architecture is said to have two queues like Read/Write Queue and Conflict Queue. The queue will work as a normal queue (FIFO) so that there will be no starvation. The client will place their request into Read/Write Queue. Master Server then investigates (parses) the type of request whether it is read or write transaction. Every transaction is given a unique transaction number by Master Server. Let the transaction taken by Master Server be write . The write transaction is simply forwarded to all the Replica Servers attached with Master Server. Each Replica Server will check the lock bit at table's necessary records at its storage. Let lock bit=0 at all Replica Servers. They will pass the positive acknowledgement to Master along with transaction number already generated **(Phase 1)**. Master Server will simply count those acknowledgements. If the count is equal to the number of Replica Servers, Master Server will allow each Replica Server to commit the transaction **(Phase 2)**. Let Replica Servers commit the transaction and send back acknowledgement to Master Server (Figure 3). After receiving the acknowledgements, Master Server will inform about the success of transaction to the respective client and delete the write transaction from the Read/Write queue.

If any of the Replica Servers unable to send "agree" message during Phase 1, the Master Server will delete the write transaction from Read/Write Queue and insert the same write transaction to Conflict Queue.

If the Master takes a read transaction for execution, then it will find out the least-load Replica Server from the statistics maintained by it. It routes the read transaction to it. If lock bit=0 at this Replica Server, the Replica Server will provide the required data to the requested client (Figure 4) and delete the read transaction from the Read/Write Queue. If the lock bit=1, move the read transaction to Conflict Queue from Read/Write Queue. Let Conflict Queue gets its chance of execution by Master Server. Let the rejected read transaction may get its turn. Now the same process of examining the lock bit continues. If lock bit=0, success will be the result else the read transaction will be marked as a failure one to the respective client and deleted from the Conflict Queue. Let the Master Server will take the some rejected write transaction from the Conflict Queue. Again the same process is continued (ie) Master will try to get agreement from all Replica Servers.

If all agree in this chance, commit else even at this try it is not possible to get the agreement, the write transaction will be treated as a timeout one and deleted from the Conflict Queue. The Master Server will inform the client about the failure of write transaction (ie) every Write/Read transaction will be given only two chances. Therefore at every time t, the Conflict Queue is said to hold the rejected transactions and the Read/Write Queue is said to have the newly entered transactions. The transactions residing at Conflict Queue will get the next chance for execution. If go through CRACS positively, it will give results to Clients. If no, it will be getting rid of the system and gives failure message to Client. It will be the wish of the clients to again place the same transaction request or not.

Journal of Theoretical and Applied Information Technology
30th April 2014. Vol. 62 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

10 SALIENT FEATURES OF CRACS

CRACS uses the familiar, simple and effective Two-Phase Commit Protocol during write transactions. CRACS does write transaction at all Replica Servers in parallel to achieve Replica Consistency and Atomicity (all or nothing property) (ie) write at all Replicas else withdraw from write. So CRACS will mostly produce the stale access rate as zero. CRACS adopts mutual exclusion principle while committing transactions and hence isolation of transactions will be maintained. Since the acknowledgement is given by Replica Server to Master Server and from Master Server to respective client, CRACS is said to be a reliable. Since CRACS follows reliable transmission protocol, assured durability of committed transactions can be given.

CRACS adopts multicast mode (a single command for a set of servers in parallel similar to group communication) for write transaction, only one write message is enough to build by Master Server towards Replicas. CRACS gives only two chances for either Read/Write transaction by considering time as an important constraint. CRACS implements row level locking to avoid lost update problem. CRACS redirects the read transaction to lightly loaded server in order to fasten the operation and therefore implements the concept of load balancing. CRACS supplies unique transaction number to each and every transaction and so it is possible to keep track of any transaction at any Replicas based on that transaction number.

11. RESULTS

Let the transaction arrives with a rate λ. Transactions interval times and exponentially distributed with mean 1/ λ. Transactions are independent identically distributed random variables, the common distribution being exponential with mean 1/μ. Let N be the number of transactions in the system (those queued plus one under service) at time t. Therefore ratio $\delta = \lambda/\mu$ = Mean Service Time / Mean Interarrival Time = Traffic Intensity. The scheduling discipline is the server is not idle with there are jobs waiting for service and any transaction is not allowed to leave the system before completion (either success/failure) with proper acknowledgement.

Assume that

E[R] = Average Response Time where R = random variable denotes the response time in steady state. Then,

$$E[N] = \delta/(1- \delta) \quad (1)$$

Little's Formula states that mean number of jobs in a queuing system in steady-state ($\delta < 1$) is equal to the product of the arrival rate and the mean response time.

So E[N] becomes λE[R] with E[R] = Average Service Time / Probability that the server is idle.

If an arriving job finds 'n' jobs in the system, then the response time is the sum of n+1 random variables, $S+S_1+S_2+\ldots+S_n$.

S = Service time of the tagged job.

S_1= Remaining service time of the job undergoing service.

$S_2 \ldots S_n$ = Service time of (n-1) jobs waiting in Queue.

Then waiting time of a transaction W = R-S.

Average number of jobs in the Queue

$$E[Q] = \lambda E[W] = \delta^2/(1- \delta) \quad (2)$$

where Q = Number of jobs waiting in the Queue.

With the above calculations, the following table (Table 1) is derived showing the execution time taken by the transactions when installing one, two, three and four replicas respectively. Four graphs (Figure 5, Figure 6, Figure 7, Figure 8) are correspondingly plotted for the execution time taken by the transactions in milliseconds. On examining the table and graph, it is vivid that the performance of the CRACS is stable even though increasing the number of replicas (ie) the execution time taken for the transactions to complete is same for both when number of replica is one or number of replicas is four. Hence the scalability of CRACS is achieved on adding the Replica Server to Master Server.

Journal of Theoretical and Applied Information Technology
30th April 2014. Vol. 62 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Execution time taken by Transactions

No. TXN	Replica 1	Replica 2	Replica 3	Replica 4
1	62.7487	57.4648	59.6686	63.1661
5	111.8385	111.5655	111.4233	113.9552
10	174.0571	173.7273	173.7529	179.0481
15	235.9202	235.7692	235.8957	235.9449
20	301.9848	297.8406	297.6283	302.9761
25	359.9003	359.7616	359.9606	359.6677
30	421.9396	422.0640	421.8927	426.9336
35	483.8847	483.4632	483.5447	483.7208
40	546.0340	545.9217	551.0043	545.9421
45	607.8006	628.5857	607.9370	628.4162
50	669.8896	669.7349	669.5836	674.9845
55	731.8993	731.7923	743.4027	747.3304
60	799.0920	793.7767	792.6075	793.8123
65	855.8728	855.9129	861.1311	855.8346
70	917.9254	922.8865	985.3209	922.9373
75	979.8270	979.6737	979.6462	979.8114
80	1041.8951	1041.9499	1041.8320	1046.9093
85	1103.8182	1103.5851	1103.7922	1119.3741
90	1165.8605	1165.8658	1170.9717	1166.2597
95	1227.8490	1227.5031	1227.6718	1237.6595
100	1289.8745	1289.6114	1289.8046	1320.7453
125	1600.7832	1599.7592	1599.7898	1604.9348
150	1909.7204	1909.8560	1914.9855	1909.7202
175	2219.6866	2219.5474	2219.7188	2235.2881
200	2529.8716	2529.7485	2529.6513	2555.7383
225	2839.4651	2839.7812	2839.7630	2855.3124
250	3175.5839	3149.5080	3149.6643	3159.8831
275	3464.8120	3464.7107	3459.6435	3490.7756
300	3779.9423	3773.6642	3779.9776	3785.0923

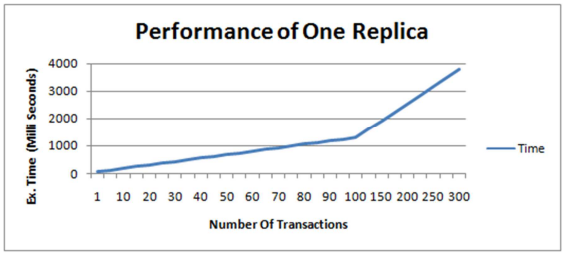

Figure 5: One Replica vs 300 Transactions

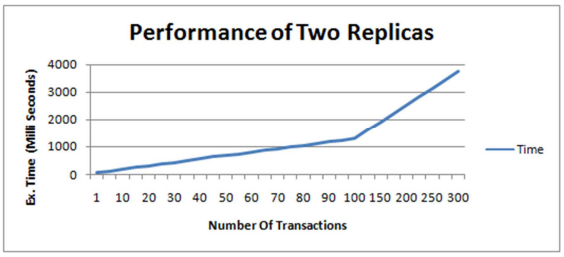

Figure 6: Two Replicas vs 300 Transactions

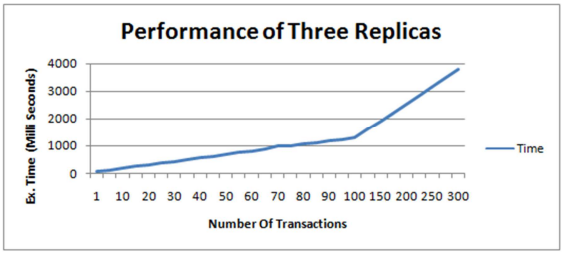

Figure 7: Three Replicas vs 300 Transactions

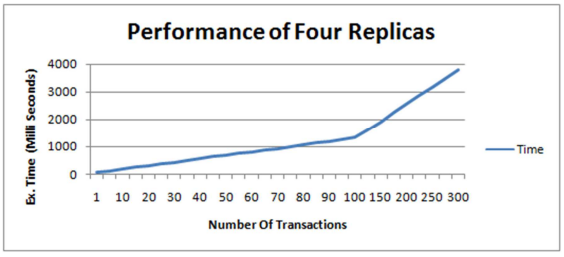

Figure 8: Four Replicas vs 300 Transactions

12. SAMPLE SCREEN SHOTS

Figure 9: Activation of Primary Server

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

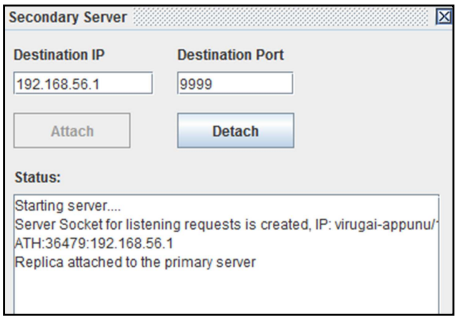

Figure 10: Attachment of Replica to Primary Server

13. SAMPLE CODE

```
 // Starting Primary Server
 new StartServer();
 // Starting Read/Write Queue Handler
 qh = new QueueHandler();
 qh.setPriority(10);
 qh.start();
 // Starting Conflict Queue Handler
 cqh = new ConflictHandler();
 cqh.setPriority(9);
 cqh.start();
 // Timer to update the system resources from
available replica periodically
   if (debugmode == 1)
System.out.println("Before
          starting timer");
           mytimer = new Timer();
           mytimer.scheduleAtFixedRate
                 (new TimerHandle(),1000,50000);
            …
            …
           inFromClient = new BufferedReader
           (new InputStreamReader
            (connectionSocket.getInputStream()));
            outToClient = new DataOutputStream
           (connectionSocket.getOutputStream());
             clientQuery =
             inFromClient.readLine();
           inputtype = clientQuery.substring(0, 3);
         …
         …
         while(rdwrQueue.size() > 0)
         {
             MyQueue result = rdwrQueue.poll();
         try
         {
               QueueClient qc =
                     new QueueClient(result);
               qc.start();
          }
          catch(IOException e)
          {
             System.out.println(
              "Error creation  QueueClient: "+e);
          }
          }
```

14. CONCLUSION AND FUTURE SCOPE

As per the above discussion, our system design CRACS helps to achieve ACID at NOSQL databases. Considering some limitations, since Master Server maintains the entire functionalities of CRACS, it will be bottleneck to the Master Server. If the Master Server is down, it will critical for CRACS for carrying out the functions. Moreover all Replica Servers are assumed to be up at all times. If at least one Replica Server is down after passing agreement message to Master Server (ie) after Phase 1, consistency becomes hard in CRACS. All transactions are assumed to be simple producing no sub-transactions. So the future scopes of this proposal are

A. CRACS will be improving the fault tolerance at both Master Server and Replica Server.

B. Moreover the Master Server is build to process even sub-transactions if any.

C. Instead of deleting the Read Transaction if the lightly-loaded server is unavailable, we can try at the second lightly-loaded server in order to make the Read Transaction a successful one.

REFERENCES

[1] "*Cloud computing — Issues, research and implementations, Information Technology Interfaces*", by Mladen A. Vouk at 30th International Conference (ITI 2008), 2008, pp. 31–40. (Conference Proceedings)

[2] "A Trust Evaluation Model for QoS Guarantee in Cloud Systems" by Hyukho Kim, Hana Lee, Woongsup Kim, Yangwoo Kim in *International Journal of Grid and Distributed Computing Vol.3*, No.1, March, 2010.

[3] "A Challenge in Improving the Consistency of Transactions in Cloud Databases - Scalability" by R. Anandhi, Dr. K. Chitra in *International Journal of Computer Applications (0975 – 8887) Volume 52*– No.2, August 2012

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] "An Overview about Cloud Computing" by R.Anandhi and Dr. K. Chitra in *International Journal of Information Technology and Knowledge Management, January-June 2012, Volume 5, No. 1*, pp. 27-30

[5] http://en.wikipedia.org/wiki/Cloud_database

[6] http://en.wikipedia.org/wiki/Database_transaction

[7] http://www.cs.helsinki.fi/group/cinco/teaching/2009/advanced-businesstransactions-seminar/papers/ACID_in_Distributed_Database_Shiwei_Yu.pdf

[8] "*Towards Next Generation Data Center Architecture: Scalability and Commoditization*" by Albert Greenberg, Parantap Lahiri, David A. Maltz, Parveen Patel, Sudipta Sengupta Microsoft Research, Redmond, WA, USA.

[9] http://www.couchbase.com/why-nosql/nosql-database

[10] http://stackoverflow.com/questions/3342497/explanation-of-base-terminology

[11] "A Study about Comparison on Consistency Models in Cloud Transactions" by R. Anandhi, G. Sekar, N.M. Elango, and Dr. K. Chitra in *IJCSSEECE 3(2) 2012*, pp 35-4.

[12] "*PBS at Work: Advancing Data Management with Consistency Metrics*" by Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, Ion Stoica UC Berkeley.

[13] "Consistency Management of Replicas in Wireless Grid Environment" by Ghalem Belalem, Lamia Allal, Cherifa in *International Journal of Grid and Distributed Computing Vol. 3, No. 4*, December, 2010. (Journal)

[14] "Developing an Enterprise Cloud Computing Strategy" by Hong Li, Jeff Sedayao, Jay Hahn-Steichen, Ed Jimison, Catherine Spence and Sudip Chahal in *Korea Information Processing Society Review, Volume 16*, Issue 2, 2009, pp. 4-16. (Journal)