ISSN:0975-9646

Scalable Transactions for Web Applications in the Cloud using Customized CloudTPS

Shashikant Mahadu Bankar

Department of Computer science and Engineering GECA,
Dr Babasaheb Ambedkar Marathwada university Auranagabad, India

***Abstract*--Data consistency is big issue while using NoSQL Cloud data stores. They ensure scalability and high availability properties for web applications, but while providing these they sacrifice data consistency. Some available applications cannot afford data inconsistency. To achieve Data consistency in multi-item transactions on web applications, CloudTPS is best solution. CloudTPS acts as a scalable transaction manager which guarantees full ACID properties for multi-item transactions on web applications. It does not depend on the presence of server failures and network partitions. There is no effect of failures and network partitions on functionality of CloudTPS. HBase and Hadoop provides scalable data layers. Hence we perform this approach on top of this scalable data layers.**

***Keywords*—Scalability, Web applications, cloud computing, transactions, NoSQL.**

I. INTRODUCTION

HBase and Hadoop are NoSQL cloud database services which provide a scalable data tier for applications deployed in the cloud. These available systems partition the application data to provide additional scalability and reproduce the partitioned data to tolerate server failures [1]. Cloud computing provides vision of a virtually infinite pool of computing, storage and networking resources, in which we can deploy scalable applications.

A transaction is a set of queries to be executed on a single consistent view of a database. The main challenge for transactions is to provide the ACID properties of Atomicity, Consistency, Isolation and Durability without negotiating the scalability properties of the cloud. However, the elemental cloud data storage services provide only conditional consistency [1].

Any centralized transaction manager would look at two scalability problems: 1) A single transaction manager must execute all incoming transactions and would finally become the performance and availability barrier; 2) A single transaction manager must control a copy of all data accessed by transactions and would finally run out of storage space. To support scalable transactions, we propose to split the transaction manager into any number of Local Transaction Managers (LTMs) and to partition the application data and the load of transaction processing across LTMs [2].

CloudTPS adventure three properties of Web applications to allow efficient and scalable operations. First, we observe that in Web applications, all transactions are short-term because each transaction is covered in the processing of a particular request from a user. Second, Web applications contribute to issue transactions that interval a relatively small number of well-identified data items. This means that the commit protocol for any given transaction can be restricted to a relatively small number of servers holding the accessed data items. Third, many read-only queries of Web applications can produce useful results by accessing an older still persistent version of data. This allows execution of complex read queries directly in the cloud data service, rather than in LTMs.

We must have to consider two important issues to handle CloudTPS conveniently:

1) There is large availability of different types of cloud services. CloudTPS must have to be portable across available cloud services. Current cloud data services use different data models and interfaces but proposed system constructs CloudTPS depending on their common features. Our method is implemented using key-value pairs. Our implementation claims a simple primary-key-based "GET/PUT" interface from cloud data services.

2) Loading of a whole copy of application into systems memory may overflow memory of LTM's. This will result into one application may use several LTMs according to their storage capacity. This is not necessary condition that only latest accessed items maintain ACID properties. If we retrieve current stored versions of unaccessed data items, then they can be ejected from the LTMs. Web applications describes temporary locations where only some portion of data is actually accessed at any time. To ensure robust data consistency, we can construct active memory management scheme to reduce the number of in-memory data items in LTMs [1].

CloudTPS must have to maintain the ACID properties even in the case of server breakdowns. For this, we reproduce data items and transaction states to multiple LTMs, and annually checkpoint consistent data snapshots to the cloud storage service. Consistency correctness depends on the final consistency and high availability properties of Cloud computing storage services [3].CloudTPS supports both read-write and read-only transactions.We check out our prototype in a workload developed from the TPC-W e-commerce benchmark [9]. We applied CloudTPS on the top of Hbase and Hadoop, which is scalable data layer. CloudTPS tolerates server break downs, which results into a few aborted transactions and a temporary decrease in throughput while transaction recovery and data reorganization. Dealing with network separations, CloudTPS may refuse incoming transactions to manage data consistency. As soon as, the network is rebuilds, transactions are recovered and becomes available.

II. RELATED WORK

A. Data Storage

Simplest technique to store structured data in to cloud is to deploy a relational database such as MySQL or Oracle. The relational data model enforced through the SQL language, results into great flexibility in accessing data. It supports practical data access operations such as aggregation, range queries, join queries, etc. Person can efficiently deploy a classical RDBMS in the cloud and thus get support for transactional consistency. These flexible query language and robust data consistency avoids partitioning of data, which introduce scalability. These systems depends on reproduction techniques and therefore do not deliver extra scalability improvement compared to a non-cloud deployment.

Some cloud database services as Bigtable, SimpleDB, uses abridged data models based consisting attribute-value pairs. All application data is arranger into tables. Data items of the tables are generally accessed through "GET/ PUT" interface. All operations are restricted to performed within table, none of them supports operations across multiple tables such as join queries. These system allows any number of tables to separate application data [5].

B. Distributed Transactional Systems

Large number of research efforts have been actively applying to distributed transactions for distributed database systems. Different types of commit protocols and concurrency control mechanisms are invented to cultivate the ACID properties of distributed transactions. Still, some distributed database make use of RDBMS. They lack in scalability as they are unable to separate application data automatically. But we can use 2-Phase Commit (2PC) for assuring Atomicity and on timestamp-ordering to maintain concurrency control.

H-Store is a distributed main memory OLTP database. It supports transactions accessing multiple data records with SQL semantics, applied as predefined stored procedures. It reproduce data records to tolerate machine failures. H-Stores scalability depends on the data separation across executor nodes. H-Store does not maintain constant logs or keep any data in the non-volatile storage of either the executor nodes or any backing store. CloudTPS checkpoints return updates back to the cloud data services to assure durability for each transaction [2].

Another system is the Scalaris transactional DHT system. It distribute data across any number of DHT nodes. It provides access to any data items by using primary key. It do not support durability for stored data as it is purely an in-memory system. CloudTPS results into durability for transactions by check pointing data updates into the cloud data service. Scalaris depends on the Paxos transactional algorithm, which can address Byzantine failures, but results into high costs for each transaction.

Google Percolator implements multirow ACID transactions on top of Bigtable. To administrate transaction management, Percolator applies Bigtable as a shared memory for all instances of its client-side library [6]. The data updates and transaction administration information, as locks and primary node of a transaction, are straightly written into Bigtable. Percolator can atomically perform many actions on a single row using single rows transactions of Bigtable such as lock a data item and mark the primary node of the transaction. In adverse, CloudTPS continue with the data updates, transaction states and queue of transactions all in the memory of LTMs. The basic cloud data store does not participate in the transaction administration. LTMs checkpoint data updates back to the cloud data store only after the transaction has been committed. The design differences of CloudTPS and Percolator arise from their distinct focuses. CloudTPS targets response-time sensitive Web applications, while Percolator is designed for incremental processing of massive data processing tasks which typically have a relaxed latency requirement.

III. PROPOSED SYSTEM

Following figure shows the complete organization of CloudTPS.

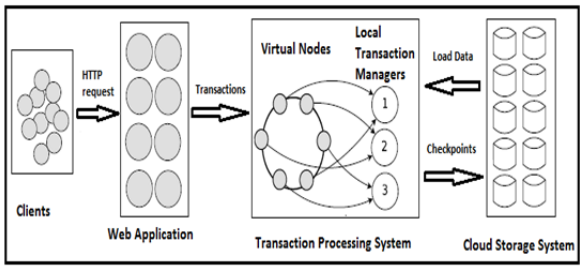

Fig 1: organization of CloudTPS system

Clients concern HTTP requests to a Web application, which consecutively concern transactions to a Transaction Processing System (TPS). The TPS be adjunct with any number of LTMs, each of which is authoritative for a subset of all data items. The Web application can submit a transaction to any one LTM that is authoritative for one of the accessed data items. This LTM then acts as the administrator of the transaction across all LTMs. Then LTM act on an inmemory copy of the entire data items which gets loaded from the cloud storage service. Updates of data transactions are placed in memory of LTMs. To avoid data loss resulting from breakdown of LTM server, the data updates are clone to multiple LTM servers. LTMs also regularly checkpoint the updates back to the cloud storage service which is considered to be highly-available and constant.

We applied transactions using the 2-Phase Commit protocol. In the very first phase, the administrator requests all involved LTMs and check whether the operation can easily executed correctly or not. If working of LTMs is proper, then second phase starts. In reality, second phase commits the transaction. Otherwise, the transaction is interrupted. Most of all cloud transactions are of short duration and can access well analyzed data items only. CloudTPS confess only server side transactions carried out as predefined procedures stored at all LTMs. Each transaction consists of one or more sub-transactions, which operate on a single data item each. When it issues a transaction, the application must provide the primary keys of all accessed data items.

Generally, a transaction is carried out as a Java object containing a list of sub-transaction instances. All sub-transactions are implemented as sub-classes of the Sub Transaction abstract Java class. Each sub-transaction consists of a unique class name to identify itself, a table name and primary key, input parameters. Already bytecode of all sub-transactions is deployed at all LTMs. A Web application concern a transaction by submitting the names of included sub-transactions and their parameters. LTMs then build up the corresponding sub-transaction instances to execute the transaction.

We first cluster data items into virtual nodes, and then attach virtual nodes to LTMs. This results into balanced assignment of virtual nodes to LTMs. Multiple virtual nodes can be allowed to the same LTM for transactions. To permit LTM break downs, virtual nodes and transaction states are duplicated to one or more LTMs. After the LTM server failure, the current updates can then be reborned and damaged transactions can continue execution while satisfying ACID properties.

We now structure the design of the TPS to assure the Atomicity, Consistency, Isolation and Durability properties. Each of the properties is discussed individually as follows:

1. Atomicity

When either all operations of a transactions are successfully executed or when none of them is executed then the property atomicity results out. CloudTPS carried out two-phase commit across all the LTMs which are chargeable for all data items accessed to assure atomicity for each transactions. The transaction administrator can concurrently return the result to the web application and complete the second phase, when an agreement to "COMMIT" is arrived [1].If the server break downs then all transactions states and all data items must have to reflect on one or more LTMs.LTMs reproduce the data items to keep backup of LTMs while execution of the second phase of transaction.When second phase completes execution successfully duplicates of the accessed data items are becomes consistent.

2. Consistency

The condition for consistent property is that when a transaction executes on an internally consistent database then it should leave the database in consistent stage. The term Consistency is commonly defined as a set of informative integrity constraints. So when transactions are completed correctly, the Consistency property is fulfilled [1].

3. Isolation

The isolation property results out when the behavior of a transaction is not changed by the existence of other transaction which also simultaneously access the same data items simultaneously. CloudTPS is responsible for the breaking down of the transaction into it's of sub-transactions. If two transactions accessing the same data items then their sub transactions must be executed in sequence, even if the sub-transactions are executed on multiple LTMs simultaneously. For that we use timestamp ordering to regulate transactions on LTMs. Each transaction has its universal exclusive timestamp order number. Sub transactions having lower timestamp ordering are executed first than sub transactions having younger timestamp ordering. The case may arise where processing of a transaction gets slow, and that a conflicting sub-transaction having younger timestamp has committed already. In such case, earlier transaction will interrupted, gets new timestamp order number and then starts re-execution [1].

4. Durability

Durability property arises when outcomes of the transactions cannot be accomplished and must have to exists when server breakdowns. Updates of all data of committed transactions must be written to the backend cloud storage service. Main problem is to support LTM break down without dropping data.Straightforwardly, the commit operation of a transaction does not update data in the cloud storage service but only update data in-memory, to increase performance. All data items get saved in to LTMs. Time period between commit operation of a transaction and upcoming checkpoints assures durability property by reproducing data items on different LTMs [1].

IV. RESULTS AND ANALYSIS

We perform evaluations on top of Hbase 0.20.6 and Hadoop v0.20.2. We use Tomcat Apache v6.0.41 as application server to evaluate CloudTPS performance. We expose the scalability of CloudTPS by evaluating the performance of a prototype implementationon top of two different families of scalabledata layers: HBase and Hadoop running on cloud.We demonstrate that proposed CloudTPS can conveniently reconstruct from server break down and network separation by considering throughput of CloudTPS under break downs.We also achieve scalability evaluation by calculatingthe maximum feasible throughput of the system including given number of LTMs before the constraintgets breach.

At beginning stage, we start with one LTM and 5 HBase servers and then we increase the number of LTM and HBase servers. Under certain number of EB's, we perform one round of the evaluation for 30 minutes to calculate the performance of the system. In all cases, we purposely allocate more number of HBase servers and client machines to assure performance barrier of the CloudTPS [6]. Fig 2 shows the efficient response time.

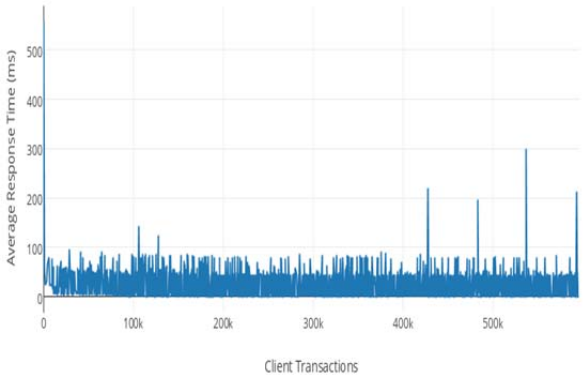

Fig 2: Graph for average Response time for Client Transactions

Performance of web server metrics depends on two things as the HTTP bytes/sec data and CPU utilization. By knowing the HTTP bytes/sec data, we can easily calculate the Mbytes/sec or Mbits/sec network traffic for each server and CPU. Consider a case where 2 processor Web Server running at 87% CPU utilization with a HTTP bytes/sec value of 4,160,450, one can calculate, (4,160,450 / (1024*1024)), a network throughput rate of 3.96 Mbytes/sec or 31.7 Mbits/sec assists by the 2P Web Server at 87% utilization. One can easily find if the web servers has any huge headroom's or web server is configured near its maximum capabilities [8].

Fig 3 and Fig 4 shows scalability illustrations by calculating throughput.

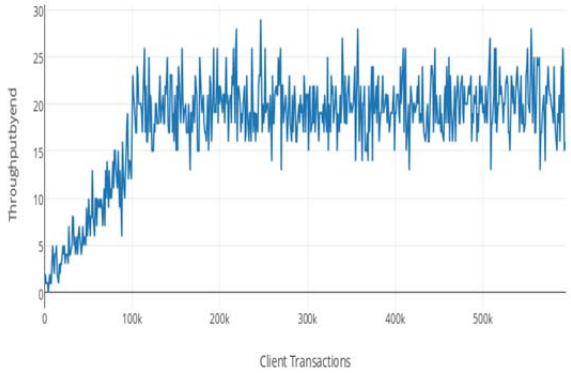

Fig 3: Graph for Total System Throughput

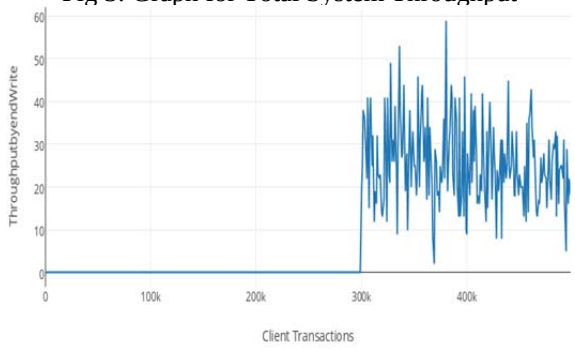

Fig 4: Graph for Throughput under Write Operation

The number of emulated users supported by each web Server is calculated by The Number of Users / Number of Web Servers. To protect the duration of the user session, the TPC-W benchmark allows keep-alive connections. Contribution of Keep-alive connections is to curtail the CPU overhead required to process a connection. Each user perceives one protected and one non-protected connection, thus we can calculate total number of connections supported by a web Server by 2 * (Number of Browsers / Web Servers). For example given a result of 4,800 WIPS with 30,000 emulated browsers in a configuration of 15 Web Servers, each Web Server is supporting 2 * (30,000 / 15)= 4,000 internet connections [9].

We can divide Web Server network traffic and keep-alive connections by total number of processors is server to get the network traffic per processor and the number of supported connections per processor. This result is very advantageous during comparison of different Web Server processors or comparison of web Servers with different number of processors. Emulated Browsers (EB's) generates data by creating and populating six tables. EB'S are the emulated browsers which is simulated to client by sending the request through http [7].Table shown below describes the performance analysis of client transactions which evaluated by Emulated Browsers.

Sr.No	Average response time	Average accessed item	Total number of transactions	Total response time	Total accessed item
updateItemInfo	1.373134328358209	1.0	67	92	67
DeleteCartLine	2.178947368421052 7	2.0	95	207	190
getShoppingCart	3.136363636363636 2	7.053030303030303	528	1656	3724
getShortOrder	30.222429906542057	11.11588785046729	535	16169	5947
RefreshCartLine	4.033707865168539	1.367977528089887 6	356	1436	487
NewShoppingCart	8.941176470588236	1.0	34	304	34
getItemAndAuther	1.372943327239488 2	2.0	547	751	1094
getOrder	18.411167512690355	7.0	591	10881	4137
getShoppingCart_inPurchase	3.205882352941176 6	9.823529411764707	34	109	334
Purchase	42.235294117647 06	18.088235 29411765	34	1436	615
getCustomer	8.837164750957854	4.0	522	4613	2088
getRelatedItem	1.058490566037736	2.0	530	561	1060
updateRelatedItemInfo	2.363636363636363 8	2.969696969696969 7	66	156	196

Table 1: Overview of client transactions for performance analysis log generated by EB'S

Following table 2 and figure 5 shows the overall cluster analysis of the proposed system.Table 2 illustrates the average access time, domain write time, time latency and time slice by considering several client transactions.

Domain Access Time	25.678391959798994 ms
Process Time	11.233855185909981 ms
Total throughput	14.44453677388901 ms
Domain Write Time	11.233855185909981 ms
Write Latency	10 ms
Time Slice	10 ms

Table 2: Cluster Analysis

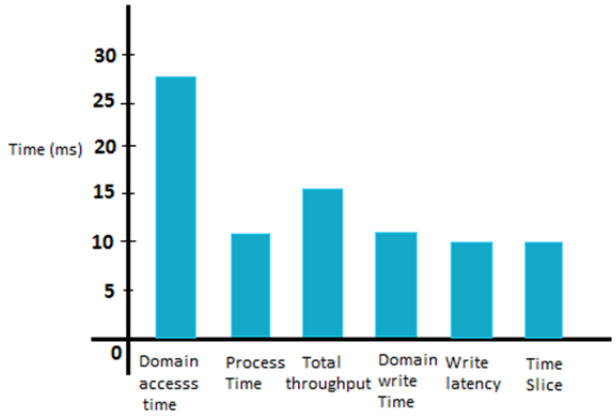

Fig 5: Cluster Analysis

V. CONCLUSION

For correct execution, products need strong data consistency. Cloud provide good platform to host web content to achieve high scalability and availability. Proposed scheme provides ACID transactions without negotiating the scalability property of the cloud for Web applications. This work depends on few simple logics. First, we load data into the transactional layer from the cloud storage system. Secondly, we can split the data across any number of LTMs, and reproduce them only for fault tolerance. Web applications can access only few partitions of data in any transactions, which results into CloudTPS linear scalability. Even in the presence of server failures and network partitions, CloudTPS supports full ACID properties. Recovering from a failure only causes a temporary drop-in throughput and a few aborted transactions. Recovering from a network partition may possibly cause temporary unavailability of CloudTPS. Data partitioning also mentioned that transactions can only access data by primary key. CloudTPS allows Web applications with strong data consistency demands to be scalable deployment in the cloud. This means Web applications in the cloud do not need to compromise consistency for scalability.

FUTURE SCOPE

Hadoop has become backbone of big data platforms but holds different, sophisticated architecture as compared to DBMS. Hadoop must have to combine with realtime extensive data collection and transmission which results into faster processing of data. Sometimes Hadoop hides some complex background while providing concise user interface which causes poor performance of system. So, we can implement advance interface similar to DBMS to enhance performance of Hadoop from each and every angle. Large-scale Hadoop cluster includes very huge number of servers which are mainly responsible for energy consumption. Hadoop should be widely deployed depending on energy efficiency. In the era of big data, the terms as privacy and security has lots of importance. The big data platform should find a good balance between enforcing data access control and facilitating data processing.

REFERENCES

[1] Zhou Wei, Guillaume Pierre, Chi-Hung Chi, "CloudTPS: Scalable Transactions for web applications in the cloud", IEEE Transactions on Services Computing, Special Issue on Cloud Computing, 2011.

[2] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51 , no. 7, pp. 9–11, Jul. 2008.

[3] Transaction Processing Performance Council, "TPC benchmark C standard specification, revision 5," December 2006, *http://www.tpc.org/tpcc/*.

[4] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[5] HBase, "An open-source, distributed, column-oriented store modeled after the Google Bigtable paper," 2006, http://hadoop. apache.org/hbase/.

[6] Amazon.com, "EC2 elastic compute cloud," 2010, *http://aws.amazon.com/ec2*.

[7] Z. Wei, G. Pierre, and C.-H. Chi, "Scalable transactions for web applications in the cloud," in *Proc. Euro-Par*, 2009.

[8] W. Vogels, "Data access patterns in the Amazon.com technology platform," in *Proc. VLDB, Keynote Speech*, 2007.

[9] D. A. Menasce, "TPC-W: A benchmark for e-commerce," *IEEE Internet Computing*, vol. 6, no. 3, 2002.

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable : a distributed storage system for structured data," in *Proc. OSDI*, 2006.

[11] S. Das, D. Agrawal, and A. E. Abbadi, "Elastras: An elastic transactional data store in the cloud," in *Proc. HotCloud*, 2009.