Transactions Management in Cloud Computing

Nesrine Ali Abd-El Azim[1], Ali Hamed El Bastawissy[2]

[1]Computer Science & information Dept., Institute of Statistical Studies & Research, Cairo, Egypt
[2]Faculty of Computers and Information, Cairo University, Cairo, Egypt
nesrine_ali79@yahoo.com

Abstract

Cloud computing has emerged as a successful paradigm for web application deployment. Economies-of-scale, elasticity, and pay-per use pricing are the biggest promises of cloud. Database management systems serving these web applications form a critical component of the cloud environment. In order to serve thousands and a variety of applications and their huge amounts of data, these database management systems must not only scale-out to clusters of commodity servers, but also be self-managing, fault-tolerant, and highly available. In this paper we survey, analyze the currently applied transaction management techniques and we propose a paradigm according to which, transaction management could be depicted and handled.

Keywords: *Cloud computing, NoSQL, CAP theorem, Multi nodes access, Consistency.*

1. Introduction

Cloud computing has emerged as an extremely successful paradigm for deploying web applications. The major reasons for the successful and widespread adoption of cloud infrastructures are scalability, elasticity, pay-per-use pricing, and economies of scale [2]. Since the majority of cloud applications are data driven, database management systems (DBMSs) is an integral technology component in the overall service architecture [3]. The reason for the propagation of DBMS, in the cloud computing space is due to the features offered by DBMSs such as: overall functionality (modeling diverse types of application using the relational model), consistency (dealing with concurrent workloads without worrying about data becoming out-of-sync), performance (both high-throughput, low-latency), and reliability (ensuring safety and persistence of data in the presence of different types of failures). In spite of this success, DBMSs are not cloud-friendly, because unlike other technology components for cloud services, such as the web servers and application servers which can easily scale from a few machines to hundreds or even thousands of machines, DBMSs cannot be scaled very easily and often become the overall system scalability bottleneck [14, 16].

A new trend has arisen that abandoned the traditional DBMSs and instead has developed new data management technologies referred to as not only SQL (NoSQL) databases. The main distinction is that in traditional DBMSs, all data within a database is treated as a "whole" and it is the responsibility of the DBMS to guarantee the consistency of the entire data. In the context of NoSQL consistency is represented in different ways such as key-values [6, 7, 12, 19] where each entity is considered an independent unit of data or information and hence can be freely moved from one machine to another. Furthermore, the atomicity of application and user accesses is guaranteed only at a single-key level. This single-key level semantics of modern applications allow data to be less correlated. As a result,

modern systems can tolerate non-availability of certain portions of data while still providing reasonable service to the rest of data. Scalability has emerged as a critical requirement in cloud computing and it is the need to ensure that the system capacity can be augmented by adding additional hardware resources whenever needed (scale out) without causing any interruption in the service.

Cloud applications deployed on top of cheap, commodity machines for which failures are common, and hence, fault-tolerance, high availability, eventual consistency, and ease of administration are essential features of data management systems in the cloud [1]. As a result, data management systems in the cloud should be able to scale out using commodity servers, be fault-tolerant, highly available, and easy to administer features which are provided by key-value stores, thus making them the preferred data management solutions in the cloud. The scalability and high availability properties of key-value stores however come at a cost. First, it allows data query only by primary key rather than join queries. Second, it only provides eventual consistency: any data update becomes visible after a finite amount of time. Despite of the weak consistency property, it is suitable for a wide range of applications. However, many other applications such as, payment services, flight reservation, online auctions, web 2.0 applications, and collaborative editing cannot afford any data inconsistency. So these applications either have to fall back to traditional databases, or to rely on various ad-hoc solutions [1, 2, 3].

It is now clear that neither Key-value stores, nor traditional databases can fit all types of applications [2, 3, 16]. As a result, there is a huge demand for a solution that can bridge the gap between scalable key-value stores (to deal with consistency issue) and traditional database systems (to overcome the scalability problem). In this paper, we state the serious effort done to bridge the data management gap following a proposed paradigm, and we also figure the future research trends in the indicated area.

Section 2 provides a survey of the CAP theorem. Section 3 proposes our more detailed Multi layers Paradigm for Cloud Transaction Management and Section 4 presents Paradigm-based Systems Classifications. Section 5 provides Paradigm-based systems transactional features and Section 6 concludes the paper and outlines some research directions.

2. The CAP Theorem

Cloud based applications need consistency, availability, and partition tolerance in order to work properly. However, CAP theorem states that any distributed system can only satisfy at most two out of three of the following properties: Consistency (all records are the same in all replicas), Availability (all replicas can accept updates or inserts), and Partition tolerance (the system still functions when distributed replicas cannot talk to each other). Therefore, when data is replicated over a wide area, the system has to choose between consistency and availability as shown in Figure 1. If the consistency (C) part of the ACID is relaxed then the system will implement various forms of weaker consistency models (e.g. eventual consistency, timeline consistency) so that all replicas do not have to agree on the same value of a data item at every moment of time (e.g. PNUTS [7], ecStore [17, 20]). If the availability (A) part is relaxed then the system will implement strict consistency (e.g. MySQL).

CAP theorem cluster all types of applications according to the three previously mentioned properties [13], however we need to analyze systems according to other characteristics such as system type, data structure type, transaction execution type, and partitioning type.

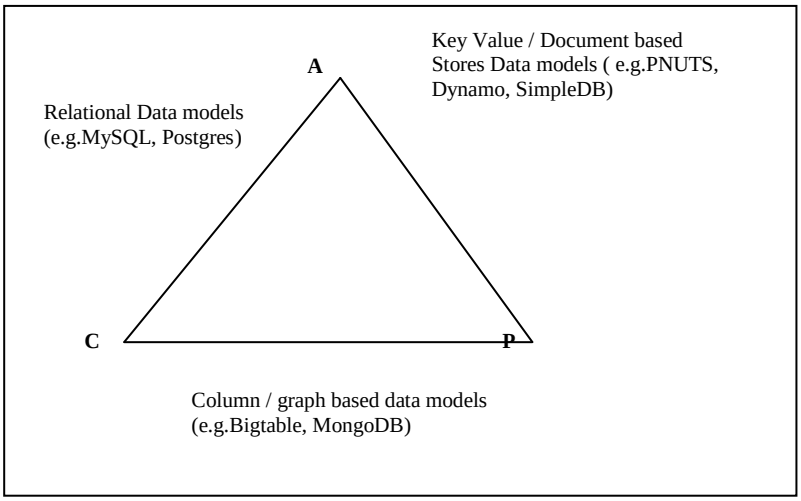

Figure 1: CAP Theorem and different data models [13]

3. Multi layers Paradigm for Cloud Transaction Management

After surveying the area of transaction management in cloud, we can suggest three layers paradigm that projects all the research trials in the area as shown in Figure 2. The first layer depicts **the system type** in which there are two types of systems: shared nothing and decoupled storage. In the shared nothing system, the persistent data is stored on disks locally attached to the nodes or DBMS servers. In the decoupled storage, the persistent data is stored in a network addressable storage accessible from all the database nodes that are logically separate from the servers executing the transactions. In decoupled storage the transaction execution logic (ACID) is decoupled from the storage logic (replication, caching, scalability, and fault tolerance).

The second layer depicts two **data structure type**: highly structure and less structured. Highly structured data stores (DBMSs) support a simple data model to store and manage data, this type have been extremely successful in classical enterprise settings due to its rich functionality, data consistency, high performance, high reliability, durability, and transactional access to data. Less structured data stores provide a mechanism for storage and retrieval of a huge amount of data that uses looser consistency models than highly structured data stores in order to achieve horizontal scaling and higher availability. They may be schema free or support simple flexible data model. They support simple functionality based on single-key operations (e.g. key value stores)

The third layer depicts **the transaction execution type** which can be classified as single node or multi nodes. In single node type, transactions can be executed at a single node without the need for distributed synchronization and the Two Phase Commit Protocol (2PC). This type of storage allows the system to scale-out by horizontally partitioning the data, it also limits the effect of a failure to only the data served by the failed component and does not

affect the operation of the remaining components, thus allowing graceful performance degradation in the event of failures. In multi node types, transactions execution cannot be contained in a single node. Therefore, the execution of a transaction could span multiple servers. Multi nodes transactions are using 2PC to permit atomic transaction commitment across the involved servers. Scalability of this type is weak due to: 1-The waiting time to get the commit from all participant servers. 2- High notification overhead. This is why we do not consider dynamic partitioning for multi nodes transactions.

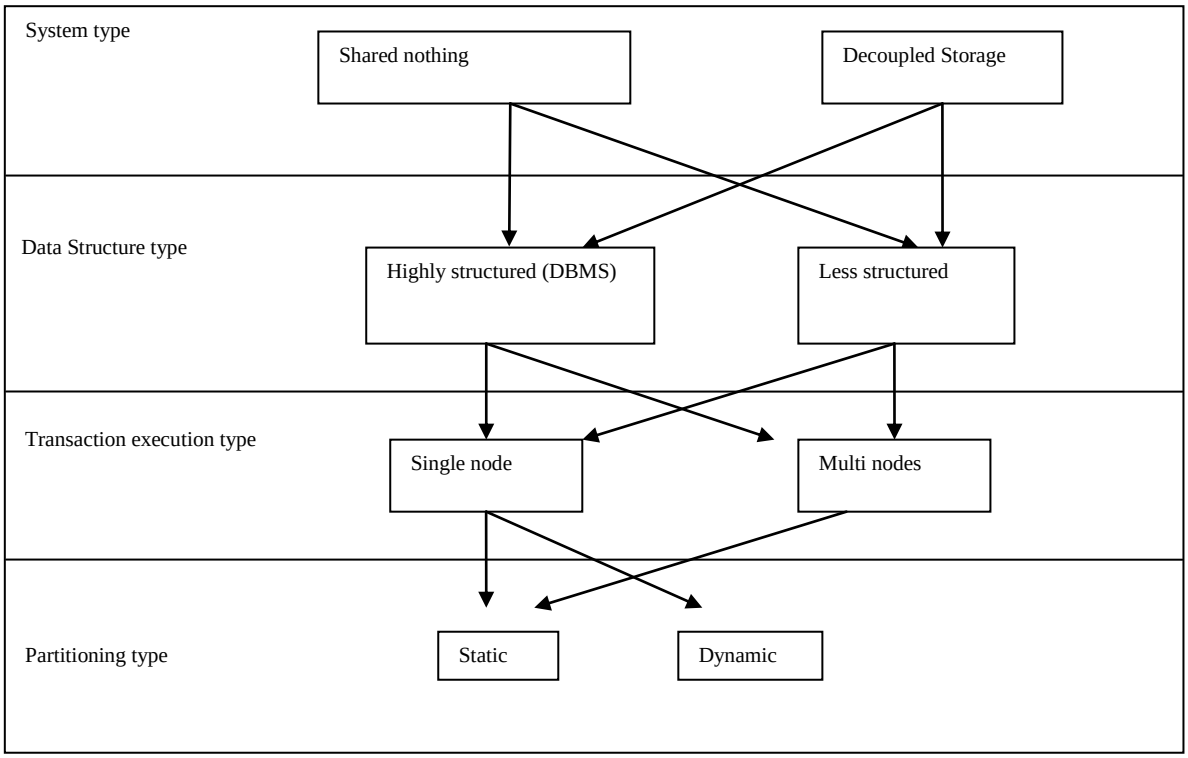

Figure 2: Multi-layers paradigm for cloud Transaction Management

The fourth layer is concerned with **the partitioning types** which are static and dynamic partitioning. Partitioning in general is a technique to scale-out a database by splitting the individual tables among a cluster of nodes and provides transactional guarantees among these nodes. Therefore, the challenge is to partition the individual tables in such way that most accesses are limited to a single partition.

In **static partitioning,** the system co-locates the relevant data items to the most frequently used application needs, in a single partition. The static partitions are then distributed among a cluster of nodes and a transaction is allowed to access one and only one cluster to guarantee scaling out efficiently. Many techniques could be applied to conform with static partitioning such as, range partitioning [17], hash partitioning [7], entity (hierarchical) group partitioning [4], and schema based partitioning [9, 10].

In **dynamic partitioning,** the system periodically analyzes the relationships between transactions and data items, upon this analysis; it relocates the data items to the nodes the most suitable for these transactions. Many techniques could be applied to conform to dynamic partitioning such as, graph based data partitioning technique [8], and key group abstraction [11].

In brief, our paradigm in Figure 2 is able to describe all transactional systems. Systems may use the classical shared nothing or the decoupled storage architecture (first layer). In both cases data can be modeled and managed using highly structured (DBMS) or less structured data stores (second layer). In both cases the transactions may be constrained to run over data on one single node or admitted to run over data distributed on multi nodes (third layer). The accessed data may be stored statistically in some node(s) or may change its locations periodically (fourth layer).

4. Paradigm-based Systems Classification

In this section we will classify the research efforts according to the previous suggested paradigm.

4.1 SHSS

The SHSS are the systems that share nothing using highly structured data store (DBMS) on single node to access data partitioned statically at this node as in [5].

4.2 SHSD

The SHSD are the systems that share nothing using highly structured data store (DBMS) on single node to access data partitioned dynamically at this node by combining a workload-aware approach for efficient data placement with a graph-based partitioning algorithm to automatically analyze the way in which transactions and data items relate to one another to automatically analyze complex query workloads and map data items to nodes to minimize the number of multi nodes transactions/statements by co-locating data items frequently accessed together in transactions as in [8].

4.3 SHMS

The SHMS are the systems that share nothing using highly structured data store (DBMS) on multi nodes to access data partitioned statically at these nodes.

4.4 SLSS

The SLSS are the systems that share nothing using less structured data store on single node to access data partitioned statically at this node as in [7].

4.5 SLSD

The SLSD are the systems that share nothing using less structured data store on single node to access data partitioned dynamically at this node.

4.6 SLMS

The SLMS are the systems that share nothing using less structured data store on multi nodes to access data partitioned statically at these nodes by using range partitioning. Range partitioning involves splitting the tables into non-overlapping ranges of their keys and then mapping the ranges to a set of nodes. It distributes tuples based on the value intervals (ranges) of some attribute. In addition to supporting exact-match queries, it is well-suited for range queries as in [17].

4.7 DHSS

The DHSS are the systems that use decoupled storage using highly structured data store (DBMS) on single node to access data partitioned statically at this node by using Schema

based Partitioning which allows designing practical and meaningful applications while being able to restrict transactional access to a single database partition. The rationale behind schema level partitioning is that in a large number of database schemas and applications, transactions only access a small number of related rows which can be potentially spread across a number of tables. This access or schema pattern can be used to group together related data into the same partition, while allowing unrelated data in different partitions, and thus limiting accesses to a single database partition as in [9, 10].

4.8 DHSD

The DHSD are the systems that use decoupled storage using highly structured data store (DBMS) on single node to access data partitioned dynamically at this node as in [15].

4.9 DHMS

The DHMS are the systems that use decoupled storage using highly structured data store (DBMS) on multi nodes to access data partitioned statically at these nodes as in [15].

4.10 DLSS

The DLSS are the systems that use decoupled storage using less structured data store on single node to access data partitioned statically at this node by using the entity group technique. An entity group is essentially a hierarchical key structure that eliminates the need for most joins by storing data that is accessed together in nearby rows or de-normalized into the same row. It consists of a root entity along with all entities in child tables that reference it as in [4].

4.11 DLSD

The DLSD are the systems that use decoupled storage using less structured data store on single node to access data partitioned dynamically at this node by using the key group abstraction. This abstraction allows applications to select members of a group from any set of keys in the data store and dynamically create (and dissolve) groups on the fly, while allowing the data store to provide efficient, scalable, and transactional access to these groups of keys. At any instant of time, a given key can participate in a single group, but during its lifetime, a key can be a member of multiple groups. Multi-key accesses are allowed only for keys that are part of a group, and only during the lifetime of the group. Groups are dynamic in nature. Groups are independent of each other and the transactions on a group guarantee consistency only within a group where transactions are not allowed across these formed groups as in [11].

4.12 DLMS

The DLMS are the systems that use decoupled storage using less structured data store on multi nodes to access data partitioned statically at these nodes by using the hash partitioning technique. In hash partitioning, the keys are hashed to the nodes serving them. It applies a hash function to some attributes that yields the partition number. This strategy allows exact-match queries on the selection attribute to be processed by exactly one node and all other queries to be processed by all the nodes in parallel as in [18].

Table 1: Cloud shared nothing systems based on our paradigm

Criteria	SHSS	SHSD	SHMS	SLSS	SLSD	SLMS
System Availability	High (due replication and dynamic quorum)	High	Fixed	High (due to geographic replication)	High	High
System Scalability	Fixed	Dynamic	Fixed	Dynamic	Dynamic	Low because no separation of system state and application state (we must consult the nodes in BATON)
System Reliability	Primary copy Replication	Replication	Replication	Geographic (asynchronous) Replication	Replication	Replication
Atomicity	Supported at single node	Supported at single node	Supported	Supported		Supported
Consistency	High consistency guarantee	High consistency guarantee	High consistency guarantee	Per record Time line	Eventual	Eventual using BASE instead of 2PC
Isolation	Serializable (locks)	Serializable	Serializable	Snapshot	Snapshot	Snapshot (MVOCC)
Durability	Log	Log	Log	Yahoo Message Broker (YMB)	Log	Log
Recovery	After image	Redo log	Redo log	Backup replicas	Redo log	Redo operations for long term failure
Applicability	Exchange Hosted Archive (EHA) and SQL Azure	Not yet functioning	Not yet functioning	Social applications	Not yet functioning	Web shop applications
Transaction Types	Non distributed short transactions	Short transactions	Short transactions	Range and exact match	Unknown	Multi keys
Load Balancing	Flexible	High (monitoring and prediction)	Fixed	Flexible	Flexible	High
Replica Contents	Static	Dynamic	Static	Static	Static	Dynamic using Two tier replication mechanism
Replica Placement	Static	Dynamic	Static	Static	Static	Dynamic using Shift key value scheme
Number of copies for each replica	Dynamic	Dynamic	Static	Dynamic	Static	Dynamic using Self-tuning range histogram
Partitioning Technique	Row group or table group	Workload aware approach with graph based partitioning	Any	Hash partitioning	Any static partitioning type	Range partitioning
CAP type	CP	CP	CP	AP	AP	AP
Systems Example	Cloud SQL Server [5]	Relational Cloud [8]	Not yet	PNUTS [7]	Not yet	ecStore [17]

Table 2: Cloud decoupled storage systems based on our paradigm

Criteria	**DHSS**	**DHSD**	**DHMS**	**DLSS**	**DLSD**	**DLMS**
System Availability	High	High as long as the transaction component (TC) is active	High as long as the transaction component (TC) is active	High	High but depends on group leader	Not high during server and network failure
System Scalability	High	Limited due to single TC	Limited due to single TC	Dynamic through partitioning	Dynamic	Linear
System Reliability	Replication	Replication	Replication	Synchronous replication	Replication	Replication
Atomicity	Supported at single node	Supported	Supported	Supported per entity group	Supported per key group	Supported (2PC)
Consistency	High	High	High	High within single entity group	High depends on underlying key value store	High
Isolation	OCC (Serializable)	Locks (serializable)	Locks (serializable)	MVCC (Serializable)	OCC (Serializable)	Timestamp ordering (Serializable)
Durability	Log	Log	Log	Log	Log	Log
Recovery	Redo log	Undo and redo log	Undo and redo log	Redo log	Undo and redo log	Redo log
Applicability	Cloud data intensive applications, payment applications	Not yet functioning	Not yet functioning	Interactive online service	Collaboration based application, On line multi players	On line book store
Transaction Types	Short transactions that access single partition	Short transactions	Short transactions	Short transactions per entity group	Long transactions	1. Short transactions that access well identified data items 2. Range query for read only transactions
Load Balancing	High	Not high due to single TC	Not high due to single TC	Low	High	Linear
Replica Content	Static	Static	Static	Static	Static	Static
Replica Placement	Dynamic	Dynamic	Dynamic	Static near the partition	Dynamic	Static
Number of copies for each replica	Static	Static due to single TC	Static due to single TC	Static	Dynamic	Static
Partitioning Technique	Scheme based partitioning	Any partitioning technique	Any partitioning technique	Entity group	Key group	Consistent hashing
CAP Type	CP	CP	CA	CP	CP	CP
Systems Example	ElasTraS [9, 10]	Deuteronomy [15]	Deuteronomy [15]	Megastore [4]	G-Store [11]	CloudTPS [18]

5. Paradigm-based Systems Transactional Features

According to the preceding paradigm-based classification, we try in this section to project on the expected features of shared nothing and decoupled system classes (Table 1 and 2 respectively).

The shared nothing systems are classified (based on data structure type) into two groups (highly structure and less structured). The first group is further classified (based on transaction execution) into two sub groups, single node transactions (SHSS, SHSD) or multi nodes transactions (SHMS). Single node transactions are well suited for OLTP and Web applications which are characterized by short lived transactions/queries with little internal parallelism. The main practical difficulty facing single node transactions was in scaling for handling skewed workloads, possible solution for this problem is data partitioning in a way that minimizes multi nodes transactions. In this type availability on unreliable commodity hardware can be maintained through replication.

On the Other hand, multi nodes transactions use two phase commit (2PC). The main disadvantage facing SHMS are the blockage of the site if coordinator fails and the introduction of a large number of messages that affects the performance. In this type availability is also achieved through replication.

the second group of the shared nothing system is also classified into two sub groups based on transaction execution either single node transaction (SLSS, SLSD) or multi nodes transactions (SLMS). Single node transactions focus on system availability and scalability with weaker consistency guarantee and no serializable transactions. These systems are applicable on social applications, which require scalability, good response time for geographically dispersed users, and high availability. At the same time, they can tolerate relaxed consistency guarantees.

On the other hand, multi node transactions (SLMS) focus primarily on atomicity and durability with weaker isolation levels through optimistic multi-version concurrency control. However, update transactions are always required in accessing the primary copy of data, both in the read and writing phases (based on the assumption that users are more likely to operate on their own data and this data tends to be independent between concurrent transactions of different users so no sharing of information between users). Another disadvantage, there is no separation of system state and application state which implies limited scalability. They are suitable for applications that might use cloud range storages (i.e. a web shop which wants to store listings in a sorted order based on their timestamps).

Similarly, the Decoupled storage systems are classified (based on data structure type) into two groups (highly structure and less structured). The first group is further classified (based on transaction execution type) into two sub groups either single node transaction (DHSS, DHSD) or multi nodes transactions (DHMS). Single node transactions are well suited for DBMS applications; however, they are not suitable for collaborative applications (that require consistent access to a dynamically formed group of keys). In Single node transactions, availability is achieved by replication, while scalability is achieved through partitioning the data in a way that minimizes multi nodes transactions.

On the Other hand, multi nodes transactions use 2PC (which may affect performance as mentioned earlier) with ACID guarantee, it is suitable for OLTP applications on disjoint set of data.

The second group of decoupled system is also classified into two sub groups (based on transaction execution type) either single node transaction (DLSS, DLSD) or multi nodes transactions (DLMS). Single node transactions focus on achieving high consistency by supporting transactional access for groups of keys on a single node and hence do not need 2PC. These groups of keys (partitions) are either static or dynamic and groups are independent of each other. Transactions on a group guarantee consistency only within that group, therefore, across partition transactions have to accept weaker consistency. They are targeted to applications whose data have a key-value schema, and require scalable and transactional access to non-disjoint groups of keys, and perform a large number of operations (long transactions).

On the other hand, multi nodes transactions use two 2PC with ACID guarantee as long as all transactions are short and span a relatively small number of well-identified data items (transactions are allowed to access any number of data items by primary key, the list of primary-keys must be given before executing the transaction). Other disadvantages include the occurrence of a temporary drop in throughput as well as a few aborted transactions as a result of server failures. In addition DLMS, it is not suitable for applications that require strict consistency.

Finally, we expect to design a system regardless of its type (Shared nothing or Decoupled system) and its data structure type (either highly or less structured) with Single node transaction execution type and Dynamic data partitioning (XXSD) to enhance its transaction capability and scalability by supporting a wider class of applications that involve overlapping partitions or require queries span multiple partitions.

6. Conclusion and Future Work

Transaction management in the cloud faces many challenges this is why we made a paradigm that deals with system types, data structure types, transaction execution types, and partitioning types. In this paper, we presented the serious efforts done to bridge the data management gaps; we classified all systems in 12 types, and we expect that system of type (XXSD) is able to enhance the current transaction features in the cloud.

References:

[1] D. J. Abadi, "Data Management in the Cloud: Limitations and Opportunities". IEEE Data Eng. Bull, vol 32(1), pp. 3-12 , 2009.

[2] D. Agrawal, A. E. Abbadi, S. Das, and A. J. Elmore, "Database scalability, elasticity, and autonomy in the cloud". In Proceedings of the 16th international conference on Database systems for advanced applications, pp. 2-15, 2011.

[3] D. Agrawal, A. El Abbadi, S. Antony, , and S. Das, "Data Management Challenges in Cloud Computing Infrastructures". In DNIS, 2010.

[4] J. Baker, C. Bond, J. C. Corbett , J. J. Furman, A. Khorlin, J. Larson, , J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore: Providing scalable, highly available storage for interactive services," in Proc. CIDR, pp. 223–234, Jan 2011.

[5] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manne, L. Novik, and T. Talius, "Adapting Microsoft SQL Server for Cloud Computing," in Proc. ICDE, pp. 1255–1263, 2011.

[6] 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, pp. 15–15, 2006.

[7] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," Proc. VLDB Endow., vol(1), pp. 1277–1288, August 2008.

[8] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich, "Relational Cloud: A Database as a Service for the Cloud," in Proc. CIDR, pp. 235–240, 2011.

[9] S. Das, D. Agrawal, and El A. Abbadi, "Elastras: an elastic transactional data store in the cloud," in Proc. USENIX HotCloud, 2009.

[10] S. Das, D. Agrawal, and El A. Abbadi, "Elastras : An elastic , scalable, and self managing transactional database for the cloud," Technical Report, CS UCSB, April 2010.

[11] S. Das, D. Agrawal, and El A. Abbadi, "G-store: a scalable data store for transactional multi key access in the cloud," in Proc. ACM SoCC, pp. 163–174, 2010.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," SIGOPS Oper. Syst. Rev, vol(41), pp. 205–220, Oct.2007

[13] S. Gilbert, and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". SIGACT News, vol(33), number 2, pp.51–59. 2002.

[14] D. Kossmann, T. Kraska, and S. Loesing, , "An evaluation of alternative architectures for transaction processing in the cloud," in SIGMOD Conference, pp. 579–590, 2010.

[15] J. J. Levandoski, D. Lome, M. F. Mokbel, and K. K. Zhao, "Deuteronomy: Transaction support for cloud data", in Proc. CIDR, pp. 123–133, Jan 2011.

[16] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," IEEE Communications Surveys and Tutorials, vol13), no. 3, 2011.

[17] H. T. VO, C. Chen, and B. C. Ooi, "Towards Elastic Transactional Cloud Storage with Range Query Support", VLDB, Sept. 2010.

[18] Z. Wei, G.Pierre, and C.-H. Chi, "CloudTPS: Scalable transactions for web applications in the cloud," Services Computing, IEEE Transactions, vol(99), 2011

[19]Amazon.com. Amazon SimpleDB., 2010. http://aws.amazon.com/simpledb.

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