TRANSACTION MANAGEMENT TECHNIQUES AND PRACTICES IN CURRENT CLOUD COMPUTING ENVIRONMENTS : A SURVEY

Ahmad Waqas[1,2], Abdul Waheed Mahessar[1], Nadeem Mahmood[1,3], Zeeshan Bhatti[1], Mostafa Karbasi[1], Asadullah Shah[1]

[1]Department of Computer Science, Kulliyah of Information and Communication Technology, International Islamic University Malaysia
[2]Department of Computer Science, Sukkur Institute of Business Administration, Pakistan
[3]Department of Computer Science, University of Karachi, Pakistan

ABSTRACT

Distributed database management systems (DDBMs) have limited capacity to manage overhead of distributed transactions due to certain failures. These types of failures don't guarantee the enforcement of transactional properties and which reduces the scalability and availability of these systems. Research in this area proved that scaling out while providing consistency of transactions and guaranteeing availability in the presence of different failures in distributed computing environment is not feasible. As a result, the database community have used vertical scaling rather than horizontal scaling. Moreover changes in data access patterns resulting from a new generation of web applications with high scalability and availability guarantees weaker consistency. In this paper we have analyse that how different systems such as ElasTraS, Mstore, Sinfonia, ecStore and Gstore provide scalable transactional data access in cloud computing environment. We have also discussed transaction management (TM) in Gstore in detail and compared with other techniques and protocols used in current cloud based systems.

KEYWORDS

Distributed Database Management Systems, Transaction Management, Cloud Computing, Transactional Data Access, Data Access Patterns

1. INTRODUCTION

Transaction is a set of query instructions to be executed atomically on a single consistent view of a database [1]. The main challenge is to provide complete transaction management (TM) support in a cloud computing environment [2]. This is achieved by implementing ACID (Atomicity, Consistency, Isolation and Durability) [1] properties without compromising the scalability properties of the cloud. However, the underlying data storage services provide mostly eventual consistency. The objective of this study is to analyse, investigate and review the key concepts and techniques with respect to TM. Research efforts in historical perspective reveal specific trends and developments in the area of TM [3][4]. Therefore, we have surveyed important research literature of the current cloud based systems and TM in DDBMSs. Our work provides a comprehensive and structured overview of TM techniques in cloud computing environment [5][6]. We have discussed following products in this paper:

i. G-Store: [7] A scalable data store for transactional multi key access in the cloud
ii. Megastore: [8] Providing Scalable, highly available storage for interactive services
iii. Scalable Transactions for web applications in the cloud [9]
iv. ElasTranS: [10]An elastic transactional data store in the cloud
v. ecStore: [11] Towards elastic transactional cloud storage with range query support

DOI : 10.5121/ijdms.2015.7104

vi. Sinfonia: [12] a new paradigm for building scalable distributed systems
vii. PNUTS: [13] Yahoo's hosted data serving platform
viii. Dynamo: [14] Amazon's highly available key-value store
ix. Bigtable: [15] A distributed storage system for structured data

Researchers have realized that supporting distributed transactions does not allow scalable and available designs [4]. Therefore to satisfy the scalability requirements web applications, designers have scarified the ability to support distributed transactions. This resulted in the design of simpler data stores based on the key-value schema, where tables are viewed as a huge collection of key-value entries. Key-value stores such as Bigtable, PNUTS, Dynamo, ecStore and their open source analogous, have been the preferred data stores for applications in the cloud. The property common to all systems is the key-value abstraction where data is viewed as key-value pairs and atomic access is supported only at the granularity of single keys. These systems limit access granularity to single key accesses, while providing minimal consistency and atomicity guarantees on multi-key accesses. While this property works well for current applications, it is insufficient for the next generation web applications which emphasize collaboration. Since collaboration by definition requires consistent access to groups of keys, scalable and consistent multi key access is critical for such applications [9].

Further the concept of key-value stores and accesses at the granularity of single keys was put forward as the sole means to attain high scalability and availability in such systems. Based on these principles, a number of key value stores also called row stores like Bigtable, PNUTS, Dynamo, ecStore and Hbase were designed and successfully implemented. Single key atomic access semantics naturally allows efficient horizontal data partitioning, and provide the basis for scalability and availability in these systems. However, all these key value stores although highly scalable, stop short of providing transactional guarantees even on a single row. Therefore to satisfy the scalability requirements of web applications, designers have scarified the ability to support distributed transactions.

As we know relational database technologies have been extremely successful in the traditional enterprise setting but they have limitations in providing additional key features and multi-key value support in cloud data management [16]. Also deployment of modern application ideas which were not technically and economically feasible in the traditional enterprise setting, have become possible due to the cloud computing paradigm.

In literature, there are important contributions in the area of database integration in cloud environment. Das [4] presented a good analysis and survey of scalable and elastic transactional data in Cloud. Sakr [17] contributed a very good and detail survey of large scale data management techniques and approaches in current cloud system. Abadi [18] highlighted important issues and challenges in managing big data in cloud computing platform. Other important publications in this regard are [16][19][9][20].

The paper is divided into six sections, after introduction in section 2 we have discussed the important concepts related to transaction management followed by section 3 which describes the cloud computing environment. Section 4 elaborates the distributed database applications followed by section 5 which describes the transaction management techniques used in distributed systems. Section 6 presents the comparison of different transaction management techniques used in distributed and cloud computing environment followed by conclusion.

2. OVERVIEW OF TRANSACTION MANAGEMENT IN TRADITIONAL DISTRIBUTED DBMS

2.1. Transaction Management

Transaction Management deals with the problems of always keeping the database in consistent state even when concurrent accesses and failures occur (Figure 1). A transaction is a collection of actions that make consistent transformations of system states while preserving system consistency [1]. In contrast a distributed transaction is a transaction that updates data on two or more networked computer systems. Distributed transactions extend the benefits of transactions to applications that must update distributed data. The consistency and reliability aspects of transactions are due to four following properties called ACID (Atomicity, Consistency, Isolation, and Durability) [1]. Atomicity ensures complete transaction execution, consistency in transaction correctness, isolation ensures transaction independence in parallel executions and durability refers to transaction persistence in database. Therefore to ensure integrity of the data, we require that the database system maintain the ACID properties of the transactions.

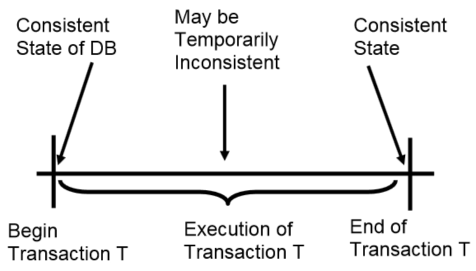


Figure 1: A transactional model

2.2. Types of Transactions

Transactions have been classified in different ways. One criterion is the duration of transactions other is the type of transaction. Gray [21] classified the transactions as online (short-life) and batch (long-life). Online transactions have a very short execution/response times and relatively affects the small portion of database. This class of transactions probably covers a large majority of current transaction applications. Banking transactions and airline reservation transactions are very good examples of online transactions. In contrast, batch transactions take longer time to execute and have access a large portion of the database. Statistical applications, report generations and image processing are the examples of batch transactions.

Transactions are also classified in terms of transaction structure [1]. Flat transaction consists of a sequence of primitives embraced between a “begin” and “end” markers and nested transactions have transactions within the main transaction. The operations of a transaction may themselves be transactions. Transactions are also categorized in terms of its read and write actions. If the transactions are restricted so that all the read actions are performed before any write actions, the transaction is called a two-step transaction. Similarly, if the transaction is restricted so that a data item has to be read before it can be updated (written), the transaction is called restricted (or read-before-write). If a transaction is both two-step and restricted it is called restricted two-step transaction.

2.3. Transaction Failures

There are different types of transaction failures. Failure can be due to an error in the transaction caused by incorrect input data as well as the detection of a present or potential deadlock [21]. Furthermore, some concurrency control algorithms do not permit a transaction to proceed or even to wait if the data that they attempt to access are currently being accessed by another transaction. The usual approach to take in cases of transaction failure is to abort the transaction, thus resetting the database to its state prior to the start of this transaction.

2.4. Database Log

Each update transaction not only changes the database but the change is also recorded in the database log. The log contains information necessary to recover the database state following a failure.

2.5. Write-Ahead Logging (WAL) Protocol

WAL protocol [22] is considered to be an important protocol for maintaining logs, whether the log is written synchronously or asynchronously. Consider a case where the updates to the database are written into the stable storage before the log is modified in stable storage to reflect the update. If a failure occurs before the log is written, the database will remain in updated form, but the log will not indicate the update that makes it impossible to recover the database to a consistent and up-to-date state. Therefore, the stable log is always updated prior to the stable database update.

We can say precisely as

1) Before a stable database is updated (perhaps due to actions of a yet uncommitted transaction), the before images should be stored in the stable log. This facilitates “undo”.
2) When a transaction commits, the after images have to be stored in the stable log prior to the updating of the stable database. This facilitates “redo”.

2.6. Check-point

In most of the local recovery manager (LRM) implementation strategies, the execution of the recovery action requires searching the entire log [21]. This is a significant overhead because the LRM is trying to find all the transactions that need to be undone and redone. The overhead can be reduced if it is possible to build a wall which signifies that the database at that point is up-to -date and consistent. In that case, the redo has to start from that point on and the undo only has to go back to that point. This process of building the wall is called check pointing. It is achieved in three steps:

1) Write a begin_checkpoint record into the log.
2) Collect the checkpoint data into the stable storage.
3) Write an end_checkpoint record into the log.

2.7. Distributed Reliability Protocols

To understand distributed reliability protocols (DRP) [1] we assume that at the originating site of a transaction there is a coordinator process and at each site where the transaction executes there are participant processes. Thus, the DRP are implemented between the coordinator and the participants. The reliability techniques in traditional DDBMS consist of commit, termination and recovery protocols. The primary requirement of commit protocols is that they maintain the atomicity of distributed transactions. This means that even though the execution of the distributed transaction involves multiple sites, some of which might fail while executing, the effects of the transaction on the distributed database is either all or nothing. This is called atomic commitment.

Also the termination protocols preferably are non-blocking which means, it permits a transaction to terminate at the operational sites without waiting for recovery of the failed site. That would significantly improve the response time performance of transactions. Moreover the DRP must be independent i.e. they have the ability to terminate a transaction that was executing at the time of a failure without having to consult any other site.

2.8. Two-Phase Commit

Two-phase commit (2PC) [21] is a very simple and elegant protocol that ensures the atomic commitment of distributed transactions. Two rules govern the decision of commit and abort, which together are called the global commit rule.

1) If even one participant votes to abort the transaction, the coordinator has to reach a global abort decision.
2) If all the participants vote to commit the transaction, the coordinator has to reach a global commit decision.

2PC permits a participant to unilateral abort a transaction until it has decided to register an affirmative vote. Once participant votes to commit or abort a transaction, it cannot change its vote. While a participant is in the ready state, it can move either to abort the transaction or to commit it, depending on the nature of the message from the coordinator. The global termination decision is taken by the coordinator according to the global commit rule. The coordinator and participant processes enter certain states where they have to wait for messages from one another. To guarantee that they can exit from these states and terminate, timers are used.

2.9. Distributed 2PL Protocol

Distributed 2PL [21] requires the availability of lock managers at each site. The distributed 2PL TM protocol is similar to the C2PL-TM [21] with two major modifications. The messages that are sent to the central site lock manager in C2PL-TM are sent to the lock managers at all participating sites in D2PL-TM.The second difference is that the operations are not passed to the data processors by the coordinating transaction manager, but by the participating lock managers. This means that the coordinating transaction manager does not wait for a lock request granted message. Also the participating data processors send the end of operation messages to the coordinating TM or each data processor can send it to its own lock manager who can then release the locks and inform the coordinating TM (Figure 2).

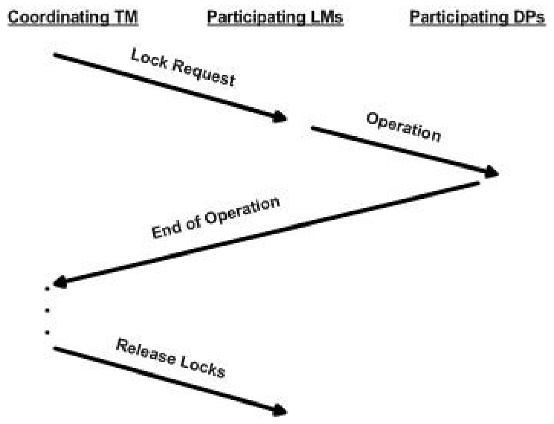


Figure 2: Communication Structure of D2PL

2.10. Optimistic Concurrency Control (OOC) Protocol

In relational databases the, OCC protocol [23] is a type of concurrency control method which assumes that multiple transactions can complete without affecting each other. Therefore transactions can proceed without locking the data resources that they affect. Before committing, each transaction verifies that no other transaction has modified its data. If the check reveals conflicting modifications, the committing transaction rolls back.

OCC is generally used in environments with low data contention. When conflicts are rare, transactions can complete without the expense of managing locks and without having transactions wait for other transactions' locks to clear, leading to higher throughput than other concurrency control methods. However, if conflicts happen often, the cost of repeatedly restarting transactions hurts performance significantly; other concurrency control methods have better performance under these conditions.

More specifically, OCC transactions involve these phases:

- Begin: Record a timestamp marking the transaction's beginning.
- Modify: Read and write database values.
- Validate: To check whether other transactions have modified data that this transaction has used (read or write).
- Commit/Rollback: If there is no conflict, make all changes to the actual state of the database if there is no conflict else resolve (abort transaction).

2.11. Mini-transactions

Mini transactions allow the applications to atomically access and conditionally modify data at multiple memory nodes. Mini transactions are also useful for improving performance, in many ways. First, mini-transactions allow users to batch together updates, which eliminate multiple network round-trips. Second, because of their limited scope, mini-transactions can be executed within the commit protocol. For example Sinfonia can start, execute, and commit a mini-transaction with two network round-trips. In contrast, database transactions, being more powerful and higher level, require two round-trips just to commit and additional round-trips to start and execute. Third, mini-transactions can execute in parallel with a replication scheme, providing availability with a replication scheme, providing availability with little extra latency.

For example, consider a cluster file system, one of the applications built with Sinfonia. With a mini-transaction, a host can atomically populate an inode stored in one memory node, and link this inode to a directory entry stored in another memory node, and these updates can be conditional on the inode being free (to avoid races). Like database transactions, mini-transactions hide the complexities that arise from concurrent execution and failures. We can say mini-transaction allow an application to update data in multiple memory nodes while ensuring ACID properties.

3. CLOUD COMPUTING AND DATA MANAGEMENT

Cloud computing eases the provision of on-demand computing resources by means of internet with surety of scalability, reliability and availability [24][31]. In accordance to National Institute of Standards and Technology (NIST), "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [NIST]. The cloud delivers resources to clients exploiting three fundamental services models that are Infrastructure-as-a-Service, Platform-as-a-Service, and Software-as-a-Services [25][26]. These service models are deployed as Public, Private, Community and Hybrid Cloud.

Cloud computing is a good business for organizations having large data centres to rent their resources. It provides efficient solutions for handling and analyzing big data. Cloud based databases predictably run and managed in a cloud computing environment that are deployed either as virtual machine images or Database-as-a-Service which follows SQL or NoSQL data models (Figure 3). The most common cloud based storage systems include OracleDB, IBM DB2, Ingres, NuoDB, BitCan, Hadoop, MongoDB, PNUTS, ecStore, Bigtable, GStore, MStore and

DynamoDB. Cloud based database management system ideally ensures few desired properties for efficient data analysis. These properties include efficiency, fault tolerance, ability to run in a heterogeneous environment, ability to operate on encrypted data and ability to interface with business intelligence products [18].

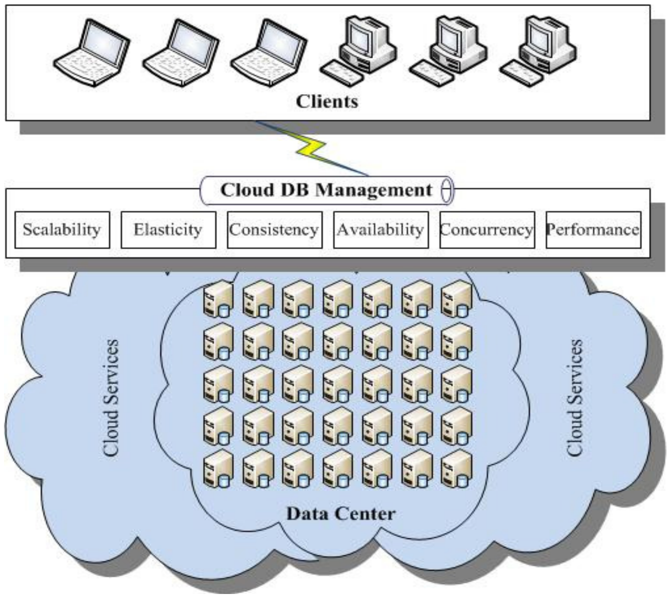


Figure 3: Communication Structure of D2PL

4. INTRODUCTION OF DISTRIBUTED STORAGE SYSTEMS

4.1. Bigtable: A distributed storage system for structured data

Bigtable is a distributed storage system for managing structured data that is designed to scale of a very large size (petabytes) of data across thousands of commodity servers. Many projects at Google store data in Big table, including Google Earth, web indexing, Orkut, and Google Finance. In many ways, Bigtable resembles a database, it shares many implementation strategies with databases and parallel databases and achieved scalability and high performance but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model instead it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Also clients can control the locality of their data through careful choices in their schemas.

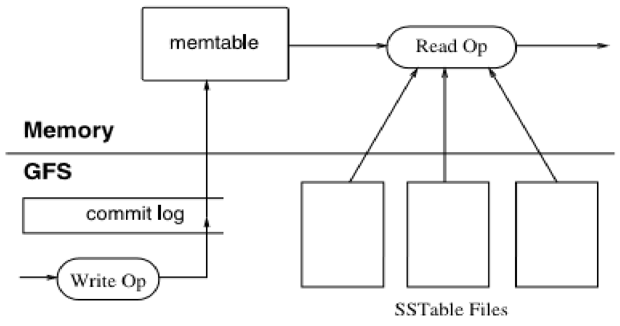


Figure 4: Bigtable Tablet

Figure 4 represents the Bigtable tablet. Big table schema parameters let clients dynamically control whether to serve data out of memory or from disk. Bigtable can be used with MapReduce [27], a frame work for running large-scale parallel computations developed at Google.

4.2. PNUTS: Yahoo!'s Hosted Data Serving Platform

PNUTS [13] is a parallel and geographically distributed database system for Yahoo's web applications. It provides data storage organized as hashed or ordered tables, low latency for large number of concurrent requests including updates and queries, and guarantees novel per-record consistency. It is a hosted, centrally managed and geographically distributed service, and utilizes automated load-balancing and fails over to reduce operational complexity.

Traditional database systems provided us a model for reasoning about consistency in the presence of concurrent operations, called serializable transactions. Generally supporting general serializable transactions over a globally replicated and distributed system is very expensive. Further, serializability of general transactions is inefficient and often unnecessary, many distributed replicated systems go to the extreme of providing only eventual consistency. In this a client can update any replica of an object and all updates to an object will eventually be applied, but potentially in different orders at different replicas.

PNUTS exposes a simple relational model to users, and supports single-table scans with predicates. Additional features include scatter-gather operations, a facility for asynchronous notification of clients and a facility for bulk loading. To meet response-times goals, PNUTS cannot use write-all replication protocols that are employed by systems deployed in localized clusters like bigtable. However, not every read of the data necessarily needs to see the most current version. Moreover it employs redundancy at multiple levels and leverages consistency model to support high-available reads and writes even after a failure or partition. PNUTS provide architecture based on record-level, asynchronous geographic replication and uses guaranteed message delivery service instead of a persistent log. PNUTS provides a consistency model that offers applications transactional features with partial serializability.

4.3. ElasTraS: An Elastic Transactional Data Store in the Cloud

ElasTras [10] is an elastic transactional data store, which was designed with the goal of scalable and elastic TM in the cloud computing environment, while providing transactional guarantees. ElasTras is analogous to partitioned databases which are common in enterprise systems, while adding features and components critical towards elasticity of the data store (Figure 5).

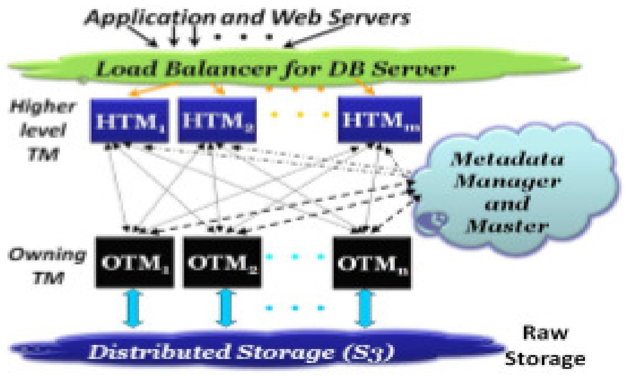
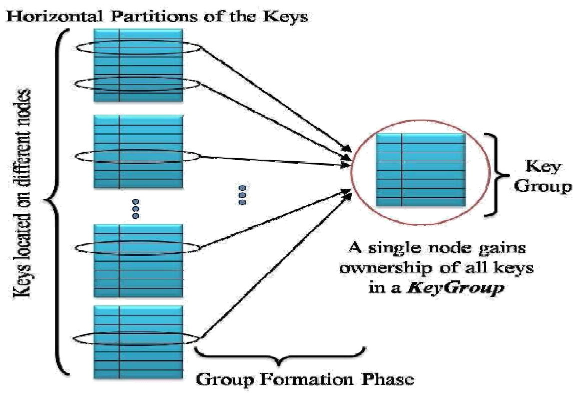


Figure 5: Overview of the ElasTraS system

It has been designed with the goal of scalable and elastic TM in the cloud. It use a key-value based design similar to Bigtable and designed with the intent to provide transactional guarantees in a scalable manner, rather than retrofitting these features into an existing systems. For providing transactional guarantees it uses two level hierarchy of TM, which are responsible for providing transactional guarantees, while also providing elastic scalability with increase in demand. ElasTras implements proven database techniques for dealing with concurrency control, isolation, and recovery, while using design principles of scalable systems (Bigtable) to overcome the limitations of DDBMS.

4.4. G-Store: A scalable Data Store for Transactional Multi key Access in the Cloud

G-Store [7] is a scalable data store providing transactional multi key access which guarantees dynamic, non-overlapping groups of keys using a key-value store as an underlying substrate. It inherits the important features of scalability, high availability and fault-tolerance. The basic innovation that allows scalable multi key access is the key group abstraction which defines a granule of on-demand transactional access. The key grouping protocol uses the key group abstraction to transfer exclusive read/write access for all keys in a group to a single node which then efficiently executes the operations on the key group. The key group abstraction allows applications to select members of group from any set of keys in the data store and dynamically create and dissolve groups, while allowing the data store to provide efficient, scalable and transactional access to these groups of keys (Figure 6). At any instant of time, a given key can participate in a single group, but during its life time, a key can be a member of multiple groups. In comparison M-Store although, supports key grouping but once a key is created as part of a group, it has to be in the group for the rest of its life time. M-Store uses the entity group abstraction for providing transactional guarantees over the collocated keys which form a contiguous part of the key range. But the static natures of entity groups as well as the contiguity requirement are often insufficient.


Horizontal Partitions of the Keys
Keys located on different nodes
Key Group
A single node gains ownership of all keys in a *KeyGroup*
Group Formation Phase
Figure 6: The key group abstraction in G-store [7]

There are no any ordering restrictions of keys in key groups therefore the members of a group can be on different nodes. Thus, using a key-value store would require distributed synchronization such as 2PC to provide atomic and consistent accesses to these keys. G-Store inherits the data model as well as the set of operations from the underlying key-value store, the only addition being that the notions of atomicity and consistency are executed from a single key to a group of keys. Moreover, it is targeted to applications whose data has an inherent key-value schema, and requires scalable and transactional access to group of keys which are formed dynamically by the applications.

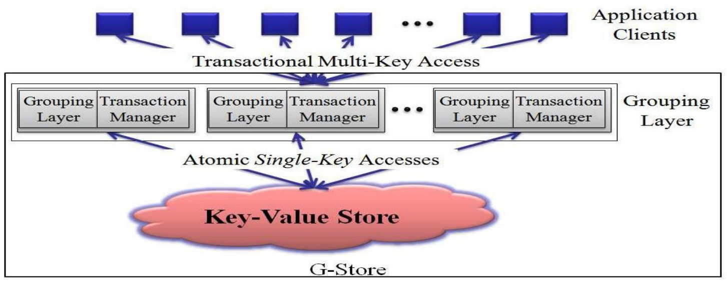


Figure 7: G-Store: Client based Implementation [7]

4.5. Scalable Transactions for Web Applications in the Cloud

It supports transaction in a scalable fashion in cloud computing environment [28]. This approach splits the transaction manager into any number of local transaction managers (LTMs) and to partitions the application data and the load of transaction processing across LTMs. Moreover, to maintain the ACID properties even in the case of server failures for transactional system they replicate data items and transaction states to multiple LTMs, and periodically checkpoint consistent data snapshots to the cloud storage service. It uses Bigtable data model, so that transactions are allowed to access any number of data items by primary-keys accessed by the transaction must be given before executing the transaction.

4.6. EcStore: Towards Elastic Transactional Cloud Storage with Range Query support

The ecStore [11] is a highly elastic distributed storage system with efficient range-query and transactional support that can be dynamically deployed in the cloud cluster. It is an elastic cloud storage system that supports automated data partitioning and replication, load balancing, efficient range query, and transactional access. In ecStore, data objects are distributed and replicated in a cluster of commodity computer nodes located in the cloud. Users can access data via transactions which bundle read and write operations on multiple data items. The storage system consists of three main layers: a distributed storage layer, a replication layer, and a TM layer.

4.7. Dynamo: Amazon's Highly Available Key-Value Store

Dynamo [14] is highly available and scalable distributed data store built for Amazons platform. It is used to manage the state of services that have very high reliability requirements and need tight control over the trade-offs between availability, consistency, cost-effectiveness and performance. It provides a simple primary key only interface to meet the requirements of these applications. In Dynamo data is partitioned and replicated using consistent hashing, and consistency is facilitated by object versioning. The consistency among replicas during updates is maintained by a quorum-like technique and a decentralized replica synchronization protocol. It employs a gossip based distributed failure detection and membership protocol. Dynamo is a completely decentralized system with minimal need for manual administration. Storage nodes can be added and removed from Dynamo without requiring any manual partitioning or redistribution. Dynamo targets applications that require only key/value access with primary focus on high availability where updates are not rejected even in the wake of network partitions or server failures. Dynamo is targeted mainly at applications that need an always 'writeable' data store where no updates are rejected due to failures or concurrent writes. Also Dynamo is built for an infrastructure within a single administrative domain where all nodes are assumed to be trusted. Moreover, Dynamo doesn’t require support for hierarchical namespaces or complex relational schema and it is built for latency sensitive applications.

4.8. Megastore: Providing Scalable, High Available Storage for Interactive Services

Megastore (M-store) [8] is a storage system developed to meet the storage requirements of modern interactive online services. Most of the commercial database systems based on relational data model have the inbuilt capacity for designing and building applications. However, they limited capacity to manage large scale applications where billions of users are involved. NoSQL [19] data stores such as Google's Bigtable [15], HBase [29] are highly scalable, but their limited API and loose consistency models complicate application development. Replicating data across distant data centres while providing low latency is challenging and it is novel in that it blends the scalability of a NoSQL data store with the convenience of relational database. It uses synchronous replication to achieve high availability and a consistent view. It provides fully serializable ACID semantics over distant replicas with low enough latencies to support interactive applications. M-store use Paxos [30] to build a highly available system that provides reasonable latencies for interactive applications while synchronously replicating writes across geographically distributed data centres. Further, M-store takes a middle ground approach in RDBMS and NoSQL design space, by partitioning the data store and replicate each partition separately, providing full ACID semantics within partitions. M-store has been widely deployed within Google for several years. It handles more than 3 billion write and 20 billion read transactions daily and stores nearly a petabyte of primary data across many global data centres.

4.9. Sinfonia: a new paradigm for building scalable distributed systems

Sinfonia [12] is a novel mini-transaction primitive that enables efficient and consistent access to data, while hiding the complexities that arise from concurrency and failures. It does not require dealing with message-passing protocols, a major complication in existing distributed systems. Moreover, protocols for handling distributed state include protocols for replication, file data and metadata management, cache consistency, and group membership. These protocols are highly non-trivial to develop. With Sinfonia developers do not have to deal with message-passing protocols. Instead, developers just design and manipulate data structures within the service Sinfonia. It targets particularly data centre infrastructure applications, such as cluster file systems, lock managers, and group communication services. These applications must be fault-tolerant and scalable, and must provide consistency and reasonable performance Figure8. In Sinfonia's mini-transactions are short-lived and streamlined to scale well in a data centre. In a nutshell, Sinfonia is a service that allows hosts to share application data in a fault-tolerant, scalable, and consistent manner.

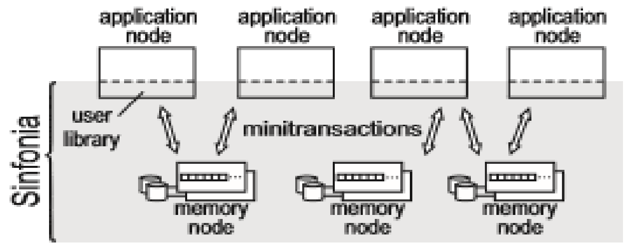


Figure 8: Sinfonia: Sharing of data in application nodes

5. Transaction Management Techniques in Distributed Systems

5.1. The Grouping Protocol

The major goal of the key grouping protocol [22] is to transfer key ownership safely from the follower to the leader during group formation, and from the leader to the followers during group deletion by satisfying the correctness and liveness properties. Group creation is initiated by an application client by sending group create request. Group formation can either be atomic or best effort. The key grouping protocol involves the leader of the group, which acts as the coordinator, and the followers, which are the owners of the keys in the group. The leader key can either be specified by the client, or is automatically selected by the system. The group create request is routed to the node which owns the leader key. The leader logs the member list, and sends a join request to all followers. Once the group creation phase completes, the client can issue operations on the group.

5.2. G-Store

G-store [7] is based on key-group abstraction which provides facility for applications to select members of a group dynamically in the data store as well as dynamically break the groups. It maintains the efficiency, scalability and transactional access to groups of keys. The key group don't have any ordering restrictions (group members can be on different nodes) compare to Mstore [8]. For TM G-Store also use the concept of key grouping protocol to ensure that the ownership of the members of a group remain within a single node. In this way implementing TM in a group does not require any distributed synchronization and is similar to TM in single node relational databases. This allows G-Store to use matured and proven techniques of database research for TM on a group. Also transactions are guaranteed to be limited to the boundaries of a single node. This design approach allows G-Store to provide efficient and scalable transactional access within a group.

In implementation of G-Store an optimistic concurrency control (OCC) is used for guaranteeing serializable transactions. Atomicity and durability properties of transactions are achieved through WAL similar to that used in databases. Also, as leader owns access to the group members, the updates are asynchronously applied to the underlying key-value store. Moreover, the key Group abstraction does not define a relationship between two groups. Groups are independent of each other and the transaction on a group generates consistency only within the confines of a group.

The key Group abstraction allows efficient execution of ACID transactions on a group while allowing the system to scale efficiently through horizontal partitioning, a design which forms the core of scalable key-value stores. G -Store is similar to RDBMSs which rely on atomic disk reads and writes at the granularity of pages to implement transactions spanning higher level logical entities such as tables, while G-Store uses atomic access at a single key granularity provided by the underlying key-value store. The only difference in G-Store is transactions are limited to smaller logical entities called groups.

During the lifetime of a group, all accesses to the members of the groups are guaranteed to be through the leader, and the leader has exclusive access to the members of the group. The leader therefore caches the contents of the group members as well as the updates made as a result of the transactions on the group. This obviates the need for expensive distributed commitment protocols like 2PC when a transaction on a group is being executed, even though the keys in a group can be physically located on multiple nodes. This results in efficient transaction execution and reduction in execution latency.

All updates to a group are appended to a WAL which is similar to a commit log in database. The leader executes the group operation, logs them, and updates its local cache which stores the data corresponding to the members of the group. The WAL is maintained as a part of the protocol state, and is separate from any log used by the underlying key-value store.

5.3. Megastore

Mstore [8] is based on entity group unlike key-groups in G-Store. In Mstore entities groups are static in nature, although it takes a step beyond single key access patterns by supporting transactional access for groups of keys, which are formed using keys with a specific hierarchical structure [8]. Since, keys cannot be updated in place, once a key is created as a part of group, it has to be in the group for the rest of its lifetime. Thus entity groups have static nature.

Entities within an entity group are mutated with single phase ACID transactions for which the commit record is replicated via paxos. In Mstore operations across entity groups could rely on expensive 2-phase commits and provide efficient asynchronous messaging. A transaction in a sending entity group places one or more messages in a queue, transactions in receiving entity groups atomically consume these messages and apply ensuing mutations. In Mstore cross entities group transactions supported via 2PC.

5.4. Scalable Transactions for Web Applications in the Cloud

In this approach the clients issues HTTP requests to a web application, which in turn issues transactions to a transaction processing system (TPS) [9]. The TPS is composed of any number of LTMs, each of which is responsible for a subset of all data items. The web application can submit a transaction to any LTM that is responsible for one of the accessed data items. This LTM then acts as the coordinator of the transaction across all LTMs in charge of the data items accessed by the transaction. The LTMs operate on an in-memory copy of the data items loaded from the cloud storage service. Data updates resulting from transactions are kept in memory of the LTMs and periodically check pointed back to the cloud storage service.

It works with the 2PL. In the first phase, the coordinator requests all involved LTMs and asks them to check that the operation can indeed been executed correctly. If all LTMs vote favourably, then the second phase actually commits the transaction, otherwise transaction is aborted.

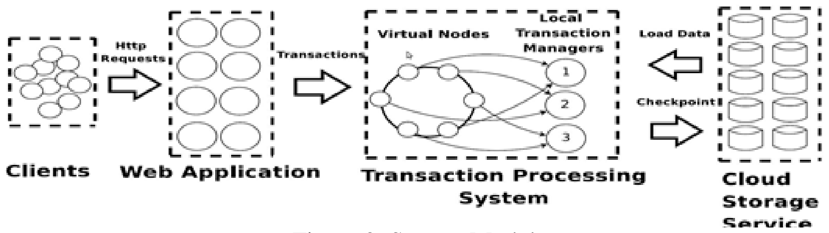


Figure 9: System Model

They assign data items to LTMs using consistent hashing. To achieve a balanced assignment, they first cluster data items into virtual nodes, and then assign virtual nodes to LTMs (Figure 9). To avoid LTM failures, virtual nodes and transaction states are replicated to one or more LTMs. After an LTM server failure, the latest updates can then be recovered and affected transactions can continue execution while satisfying ACID properties.

5.5. ElasTraS

ElasTraS [10] provides a two level hierarchy of transaction managers which are responsible for providing transactional guarantees, while providing elastic scalability with increase in demand. ElasTraS uses two types of transactional managers HTM (High level Transaction Manager) and OTM (Owning Transaction Manager).The OTM are the entities responsible for the executions on the partitions of the databases, and have exclusive access rights to the partitions they own. These are analogous to the tablet servers in Bigtable. An OTM is responsible for all the concurrency control and recovery functionality for the partitions it owns. At the top of the stack are the application and web servers that interact with the database. Requests to the database are handled through the load balancer. When a transaction request arrives, the load balancer forwards it to a HTM. The HTM decides whether it can execute the transaction locally or route it to the appropriate OTM which owns exclusive access rights to the data accessed by the transaction.

In ElasTraS the database tables are partitioned and can be configured for both static and dynamic partitioning. Static partitioning is same like database partitioning, the database designer partitions the database, and ElasTraS is responsible for mapping partitions to specific OTM and reassigning partitions with changing load characteristics to ensure scalability and elasticity. In this configuration the application is aware of the partitions, and hence can be designed to limit transactions to single partitions. In such a configuration, ElasTraS can provide ACID transactional guarantees for transactions to single partitions. In dynamic partitioning configuration, in addition to managing partition mapping, is also responsible for database partitioning using range or hash based partitioning schemes. To ensure scalability in dynamic partitioning set up, and avoid distributed transaction, it only supports mini-transactions. More precisely, TM in a statically partitioned setup, applications can limit transactions to single partitions. ElasTranS guarantees consistency only within a partition of the databases and there is no notion of consistency across partitions or global serializability.

5.6. ecStore

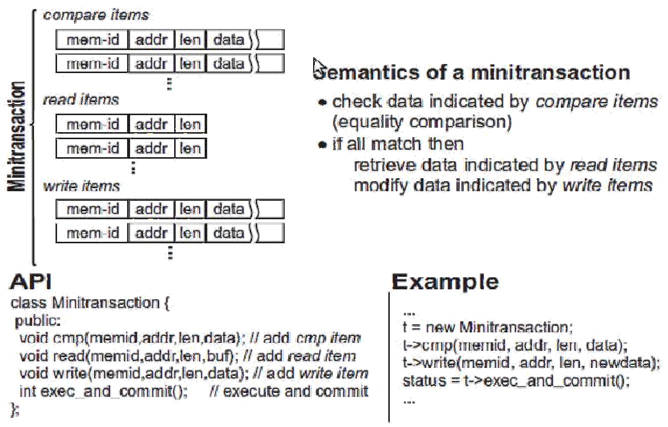
The TM in ecStore [11] use two facts or characteristics in cloud storage. First, it is usually sufficient to perform operations on a recent snapshot of data rather than on up-to-second most recent data. Second, the locality of data accessed transactions, because in web applications users are more likely to operate on their own data, which forms an entity group or a key group as characterized in Mstore and Gstore. By using both facts of cloud data, ecStore use a hybrid scheme of multi-version and optimistic concurrency control. The beauty of this approach is that multiple versions of data can benefit the read only transactions, while the optimistic method protects the system from the locking overhead of update transactions.

In this hybrid scheme, each transaction has a start-up timestamp, which is assigned when the transaction starts, and commits timestamps, which is set up during the commit process. In addition, each data object also maintains the commit timestamp of its most recent update transaction. When a transaction accesses a data object, the most recent version of the data with a timestamp less than transaction's start-up timestamp is returned. Thus no locking overhead is incurred by the read requests. However, there are two main differences when we combine with the multiversion method. First, read-only transactions run against a consistent snapshot of the database, hence they can commit without validation phase. Second, the validation phase of update transactions uses the version number of data objects to check for write-write and write-read conflicts among concurrent transactions [17]. In particular, an update transaction is allowed to commit only if the version of any data object observed by this transaction during the read phase is still the same when the transaction is validated, meaning that these data objects have not been updated by other concurrent transactions. By using version-based validation, there is no need to store old write-sets of committed transactions just for the purpose of validating read-set/write-set instructions.

5.7. Sinfonia

Mini-transactions allow an application to update in multiple memory nodes while ensuring atomicity, consistency, isolation, and durability. Mini-transactions have compare items, read items, and write items. Compare items are locations to be tested for equality against supplied data, if any test fails, the mini-transaction aborts. If the mini-transaction commits, read items are locations to be read and returned, while writes items are locations to be written (Figure 10). Application nodes execute and commit mini-transactions using the two-phase protocol. Phase 1 executes and prepares the mini-transaction, while phase 2 commits it. More precisely, in phase 1, the coordinator (application node) generates a new transaction id (tid) and sends the mini-transaction to the participants (memory nodes). First, each participant then tries to lock the address of its items in the mini-transaction (without blocking) then executes the comparisons in the compare items and, if all comparisons succeed, performs the reads in the read items and buffer the write items. Finally it decides on a vote, if all locations could be locked and all compare items succeeded, the vote is for committing, otherwise it is for aborting. In phase 2, the coordinator tells participants to commit if all votes are committing. If commit, participants applies the write items, otherwise it discards them. In either case, the participants releases all locks acquired by the mini-transaction.

The coordinator never logs any information, unlike in standard two-phase commit. If the mini-transaction aborts because some locks were busy, the coordinator retries the mini-transaction after a while using a new tid. Participants log mini-transactions in the redo-log in the first phase (if logging is enabled); logging occurs only if the participant votes to commit. Only write items are logged, not compare or read items, to save space. The redo-log in Sinfonia also serves as a write-ahead log to improve performance. Participants keep an in-doubt list of undecided transaction tids, a forced-abort list of tids that must be voted to abort, and a decided list of finished tids and their outcome.



Semantics of a minitransaction

- check data indicated by *compare items* (equality comparison)
- if all match then
  retrieve data indicated by *read items*
  modify data indicated by *write items*

```
class Minitransaction {
 public:
  void cmp(memid,addr,len,data); // add cmp item
  void read(memid,addr,len,buf); // add read item
  void write(memid,addr,len,data); // add write item
  int exec_and_commit();     // execute and commit
};
```

```
...
t = new Minitransaction;
t->cmp(memid, addr, len, data);
t->write(memid, addr, len, newdata);
status = t->exec_and_commit();
...
```

Figure 10: Sinfonia: MiniTransaction

5.8. PNUTS

PNUTS [13] uses OCC protocol for TM. In PNUTS we have test-and-set write instead of read-modify-write, where a call performs the requested write to the record if and only if the present version of the record is the same as required version. This call can be used to implement transactions that first read a record, and then do a write to the record based on the read. The test-and-set write ensures that two such concurrent increment transactions are properly serialized. The API of PNUTS with comparison to that of SQL, may be criticized for revealing too many implementation details such as sequence numbers. However, revealing these details does allow the application to indicate cases where it can do with some relaxed consistency for higher performance such as read-critical. Similarly, a test-and-set write allows us to implement single-row transactions without any locks, a highly desirable property in distributed systems. According to the need, the API can be packaged into the traditional begin transaction and commit for single-row transactions at the cost of losing expressiveness. In particular, PNUTS make no guarantees as to consistency for multi-record transactions. PNUTS model can provide serializability on a per-record basis. In particular, if an application reads or writes the same record multiple times in the same transaction, the application must use record versions to validate its own reads and writes to ensure serializability for the transaction.

6. COMPARISON OF TM IN G-STORE WITH OTHER PRODUCTS

The comparison of transaction management in G-store with other products is represented in table 1. It is very difficult to run existing shared memory applications transparently on a distributed system. The loosely coupled nature of distributed systems created hard efficiency and fault tolerance problems. Further, the G-Store depends on the underlying key-value store for consistency (table 1). If BigTable is used as underlying key-value store it provides synchronous or strong consistency and if PNUTS and Dynamo are used then it provides asynchronous consistency. Generally Mstore provides synchronous consistency but in the case of group to group communications it offers asynchronous consistency (table 1). On the other hands most of the systems use multi-version or eventual consistency.

Table 1: Comparison of Transaction Management in G-Store with other Products:

	G-Store	**MStore**	**ecStore**	**Sinfonia**	**Scalable Transaction for Web Applications in the Cloud**	**ElasTrans**
Key-Access	M-Key	Entity Groups	Multiple	Single	Single	Single
Key-Grouping	Dynamic	Static	Static	N/A	N/A	N/A
Order of Keys	NO	YES/Contiguous	YES	N/A	N/A	N/A
Concurrency Control Protocol in TM	OCC Protocol	2PC	OCC and Multiversion / Hybrid Approach	2PC	2PC	Protocols used by Traditional Distributed Databases
Partitioning	Range & Hash	Range	Range	-----	Hash	Range or Hash in Dynamic Mode
Replication	Synchronous (if Bigtable)	Synchronous (Paxos rep algo)	Asynchronous	Synchronous	Synchronous	Synchronous
Consistency	Strong (if Bigtable depends on kv store, Weak if Dy and PNUTS	Synchronous	Multiversion / Eventual	Synchronous	Multiversion / Eventual	Multiversion
Distributed Transactions	YES	YES	YES	YES	YES	YES
Appropriate for Collaborative Applications	YES	NO	NO	NO	NO	NO
Minitransactions	NO	NO	NO	YES	NO	YES in Dynamic Configuration
Atomicity and Durability	WAL	WAL	WAL	WAL	WAL	WAL

For concurrency control the G-Store uses an optimistic concurrency control protocol to guarantee the serializable transactions. In comparison, Mstore uses the expensive 2PC protocol for concurrency in transactions and ecStore use a hybrid approach of multi-version and optimistic concurrency control. The advantage of ecStore approach is the handling of multiple versions of data which benefits the read only transactions, while the optimistic method protects the system from the locking overhead of update transactions. G-Store uses the concept of key grouping protocol in TM to ensure that the ownership of the members of a group remain within a single node. In this way implementing TM in a group does not require any distributed synchronization. Moreover, G-Store is suitable for collaborative applications compared to other applications such as Sinfonia which uses distributed share memory system. The comparison of cloud based storage systems is represented in table 2.

Table 2: Comparison of Features of Cloud Based Storage Systems:

Features	Partitioning	Load Balancing	Replication		Distributed Transaction Support
			Syn/Asy	Consistency	
PNUTS	Hash/Range	Data Migration	Asyn	Time line + Eventual	No
ecStore	Range	Data Migration	Asyn	Multi-version + Eventual	Yes
Bigtable	Range	Other	Sync	Multi-version	No
Dynamo	Hash	Multi virtual nodes on a machine	Asyn	Eventual	No

7. CONCLUSIONS

There is no single database product or technology which provides the complete solution for data management challenges in cloud computing environment. There are different products (implementations) are being used to target specific database design and implementation issues in cloud environment. There is always a trade-off in using such technologies in terms of performance, availability, consistency, concurrency, scalability and elasticity. There are some important areas of further research where we still need more attention. It includes the consistency at different scales in a cloud, elastic TM techniques for elastic cloud architectures, performance management, federated databases for federated clouds, and data security in cloud environment,

We have analysed various TM techniques used in current cloud based systems along with the techniques and protocols used by traditional distributed database systems. We have focused on G-Store approach of TM and compared with other existing approaches. G-store can inherit data model from underlying key value store, so its TM technique can also be used on PNUTS, BigTable and Dynamo. The major advantage of G-Store for TM in cloud computing environment is dynamic key grouping which is the requirement of current and future collaborative web application that needs dynamic grouping without prior requirement of specific key orders. Although, Mstore also have grouping feature but that is static in nature and the key grouping technique where entity groups have specific key ordering is not appropriate for future collaborative web applications. In comparison, all other systems are based on single key access semantics. It is observed that most of the system use WAL as a preferred approach for atomicity and durability requirement of transaction.

REFERENCES

[1] P. V. Tamer Ozsu, Principles of Distributed Database Systems, Springer. 2011.

[2] B. Hayes, "Cloud computing," Commun. ACM, vol. 51, no. 7, p. 9, Jul. 2008.

[3] R. G. Tiwari, S. B. Navathe, and G. J. Kulkarni, "Towards Transactional Data Management over the Cloud," *2010 Second Int. Symp. Data, Privacy, E-Commerce*, pp. 100–106, Sep. 2010.

[4] S. Das, "Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms," 2011.

[5] A. El Abbadi, "Big Data and Cloud Computing : Current State and Future Opportunities," in *EDBT 2011*, Uppsala, Sweden, 2011, pp. 0–3.

[6] D. Kossmann, T. Kraska, and S. Loesing, "An evaluation of alternative architectures for transaction processing in the cloud," in Proceedings of the *2010 international conference on Management of data - SIGMOD* '10, 2010, pp. 579–590.

[7] S. Das and A. El Abbadi, "G-Store : A Scalable Data Store for Transactional Multi key Access in the Cloud," in *SoCC'2010*, Indianapolis, Indiana, USA, 2010, pp. 163–174.

[8] J. Baker, C. Bond, J. C. Corbett, J. J. Furman, A. Khorlin, J. Larson, L. Jean-michel, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore : Providing Scalable , Highly Available Storage for Interactive Services," in *5th Biennial Conference on Innovative Data Systems Research*, 2011, pp. 223–234.

[9] Z. Wei, G. Pierre, and C.-H. Chi, "CloudTPS: Scalable Transactions for Web Applications in the Cloud," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 525–539, 2012.

[10] S. Das, A. El Abbadi, C. Science, and U. C. S. Barbara, "ElasTraS : An Elastic Transactional Data Store in the Cloud," in *USENIX HotCloud* 2, 2009.

[11] H. T. Vo, C. Chen, and B. C. Ooi, "Towards elastic transactional cloud storage with range query support," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 506–514, Sep. 2010.

[12] M. K. Aguilera, A. Merchant, A. Veitch, and C. Karamanolis, "Sinfonia : A New Paradigm for Building Scalable Distributed Systems," in *SOSP 2007,* Stevenson, Washington, USA, 2007, pp. 159–174.

[13] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "PNUTS : Yahoo !' s Hosted Data Serving Platform," in *VLDB 2008*, Aukland, New Zealand, 2008.

[14] G. Decandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo : Amazon ' s Highly Available Key-value Store," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, 2007.

[15] F. A. Y. 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," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, 2008.

[16] D. Agrawal, A. El Abbadi, S. Antony, and S. Das, "Data Management Challenges in Cloud Computing Infrastructures," in *DNIS 2010, LNCS*, 5999, 2010, pp. 1–10.

[17] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A Survey of Large Scale Data Management Approaches in Cloud Environments," *IEEE Commun. Surv. Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.

[18] D. J. Abadi, "Data Management in the Cloud : Limitations and Opportunities," in *Bullitin of the IEEE Computer Society Technical Committee on Data Engineering*, 2009, pp. 1–10.

[19] R. Cattell, "Scalable SQL and NoSQL Data Stores," in *ACM SIGMOD Record*, 2010, vol. 39, no. 4, pp. 12–27.

[20] P. Romano and L. Rodrigues, "Cloud-TM : Harnessing the Cloud with Distributed Transactional Memories," pp. 1–6.

[21] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann. 1993.

[22] Y. Challal and H. Seba, "Group Key Management Protocols : A Novel Taxonomy," *Int. J. Inf. Technol.*, vol. 2, no. 1, pp. 105–118, 2005.

[23] X. C. Song, I. C. Society, and J. W. S. Liu, "Maintaining Temporal Consistency : Pessimistic vs . Optimistic Concurrency Control*," IEEE Trans. Knowl. Data Eng.*, vol. 7, no. 5, pp. 786–796, 1995.

[24] A. Waqas, Z. M. Yusof, A. Shah and M. A. Khan, "Architecture for Resources Sharing Between Clouds," in *2014 Conference on Information Assurance and Cyber Security (CIACS2014)*, pp.23,28, 12-13 June 2014. doi: 10.1109/CIACS.2014.6861326

[25] A. Waqas, Zulkefli.Muhammed, Asadullah Shah, "A security -based survey and classification of Cloud Architectures, State of Art and Future Directions," in *2013 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pp.284,289, 23-24 Dec. 2013. doi: 10.1109/ACSAT.2013.63

[26] Tinghuai Ma, Chu Ya, Licheng Zhao and Otgonbayar Ankhbayar ,"security -based survey and classification in Cloud Computing: policy and Algorithm," *IETE Tech. Rev.,* vol. 31, no. 1, pp. 4-16, 2014.

[27] J. Dean and S. Ghemawat, "MapReduce : Simplified Data Processing on Large Clusters," *in 6th Symposium on Operating Systems Design and Implementation*, 2004, pp. 137–149.

[28] Z. Wei, G. Pierre, and C. Chi, "Scalable Transactions for Web Applications in the Cloud," in *Euro-Par 2009, Parallel Processing,* 2009, pp. 442–453.

[29] M. N. Vora, "Hadoop-HBase for large-scale data," in *Proceedings of 2011 International Conference on Computer Science and Network Technology,* 2011, pp. 601–605.

[30] L. Lamport, "Paxos Made Simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.

[31] A. Waqas, Zulkefli.Muhammed, Asadullah Shah, "Fault Tolerant Cloud Auditing," in *2013 5th International Conference on Information and Communication Technology for the Muslim World (ICT4M)*, pp.1,5, 26-27 March 2013. doi: 10.1109/ICT4M.2013.6518924

AUTHORS

Mr Ahmad Waqas is PhD Scholar at Department of Computer Science, Faculty of Information and Communication Technology, International Islamic University Malaysia. He is working as Lecturer in the Department of Computer Science, Sukkur Institute of Business Administration Pakistan. He has been involved in teaching and research at graduate and post graduate level in the field of computer science for the last eight years. He has obtained his MCS (Masters in Computer Science) from University of Karachi in 2008 with second position in faculty. He did his MS (Computer Communication and Networks) from Sukkur IBA Pakistan. His area of interest is Distributed Computing, Cloud Computing Security and Auditing, Computing architectures, theoretical computer science, Data structure and algorithms. He has published many research papers in international journals and conference proceedings.

Mr. Abdul Waheed Mahessar is a PhD scholar at Department of Computer Science in International Islamic University Malaysia. He is working as Lecturer in Department of Information Technology at University of Sindh Jamshoro Pakistan. His area of interest includes network design and management, scale free networks, information security and data communication.

Dr. Nadeem Mahmood is working as Post-Doctoral Research fellow at Faculty of Information and Communication Technology, International Islamic University Malaysia. He is assistant professor in the Department of Computer Science, University of Karachi. He has been involved in teaching and research at graduate and post graduate level in the field of computer science for the last seventeen years. He has obtained his MCS and Ph.D. in computer science from University of Karachi in 1996 and 2010 respectively. His area of interest is temporal and fuzzy database systems, spatial database systems, artificial intelligence, knowledge management and healthcare information systems. He has published more than 20 research papers in international journals and conference proceedings (IEEE and ACM). He is working as technical and program committees' member for different international journals and conferences.

Mr. Zeeshan Bhatti is a PhD student and researcher in the discipline of Information Technology at Kulliyyah of information and Communication Technology, International Islamic University Malaysia (IIUM). His current area of research is in the field of Computer Graphics and Animation. He is specifically conducting research on generating procedural simulations of quadruped locomotion's. My research topic is titled as "Mathematically based procedural animation of quadrupeds in 3D environment". Mr. Bhatti is working as Lecturer in Department of Information Technology at Sindh University Jamshoro Pakistan. He has published many research papers in international journals and conferences.

Dr. Asadullah Shah is Professor at Department of computer science, Kulliyyah of information and communication technology, IIUM. Dr. Shah has a total of 26 years teaching and research experience. He has more than 100 research publications in International and national journals and conference proceedings. Additionaly, he authored one book and currently editing another book. Dr. Shah has done his undergraduate degree in Electronics, Master's degree in Computer Technology from the University of Sindh, and PhD in Multimedia Communication, from the University of Surrey, England, UK. His areas of interest are multimedia compression techniques, research methodologies, speech packetization and statistical multiplexing. He has been teaching courses in the fields of electronics, computers, telecommunications and management sciences.