IJRET: International Journal of Research in Engineering and Technology eISSN: 2319-1163 | pISSN: 2321-7308

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

A SURVEY ON CLOUD STORAGE – IMPACT OF STORAGE ON THE PERFORMANCE

Prapti Panigrahi[1], Jyotirmayee Rautaray[2]

[1]*BTech, Computer Science and Engineering, College of Engineering and Technology, Bhubaneswar, Odisha, India*
[2]*Sr.Lecturer, Computer Science and Engineering, College of Engineering and Technology, Bhubaneswar, Odisha, India*

Abstract

Cloud Computing refers to the idea of using computing power as a utility. It is implemented by a connected network of remote servers, accessed via the internet, for processing, storing and managing data. The cloud computing model allows for pay-by-use and thus avoids usage of personal computers or local servers. Cloud Computing is an emerging technology, gradually gaining prominence in academia as well as commercial enterprises. Storing and micromanaging data is an integral part of any application. Using cloud storage provides various benefits over traditional storage- enabling the user to remotely access/store their data anywhere and from any device, on-demand scalability of applications based on usage. In case of disasters, cloud storage can help in very quick recovery of data. Bandwidth usage can also be reduced, by sharing access-links instead of the complete files. Understanding the importance of storage mechanism and realizing that in a properly developed architecture can improve the efficiency of application by leaps and bounds. Many different types of storage architectures have been developed over time. Various commercial organizations have also come up with tailor-made storage mechanisms, suiting their specific requirements. Selecting amongst the many available storage architectures is a very complex task as well as a very important one. The storage mechanism plays a huge role in determining its overall performance. This work aims to assist the reader in proper selection of architecture based on the types of operation the user of the architecture intends to have in his/her application.

Keywords: *Cloud Storage, Virtualization, OpenStack*

---***---

1. INTRODUCTION

In 1950's, time sharing was first introduced, bringing the concept of sharing resources among applications. Later sometime in late 1960's, ARPANET was introduced that linked around four computers spread across the globe via internet. In the next few years, 'client-server' systems were developed, which allowed access of data over a local network. It was 1999, that Salesforce.com became the first company to launch a service which allowed applications to be available over a website. This form of hosting applications was known as hosting on cloud. NASA's OpenNebula [1], in 2008 was the first open-source software for various types of cloud (private, hybrid) implementation. In the last decade, many other corporate and open-source cloud platforms like Amazon Web Services, Google File Systems, Microsoft Azure, OpenStack, Apache Hadoop distribution etc. started coming up and revolutionized the way computing/storage was carried out.

All the above mentioned cloud platforms can be classified among the following three categories: private, public, or private and public (hybrid). Services such as Amazon Web Services, Windows Azure etc. are public clouds - open for the use of general public. But many organizations deploy a cloud using open source platforms like OpenStack or Hadoop, to be used for internal purposes only, known as private cloud. A hybrid cloud is used so that internal confidential data can be stored in private cloud, while other data are stored in the public section.

These services (public cloud, private cloud, hybrid cloud) have been possible because of virtualization. Virtualization is a software that manipulates hardware to provide the cloud-computing service [15]. Virtualization means creating virtual or logical version of any resource (could be operating system, server, storage etc). Storage Virtualization is a technology that enables many logical storage devices from one single physical storage. With remote access to storage devices also feasible now, a new form of enterprise-Infrastructure as a service (IaaS) has been developed. Here the client instead of investing on its own storage, can simply 'lease' the storage devices from a provider. This also allows the client to be free from any form of maintenance issues.

While this methodology works for small scale applications, many service providers prefer to develop their own storage architecture for various reasons – security worries, customizable structure for improved performance, higher flexibility, using existing hardware and much more.

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

Creating storage on the cloud – private or public, is inevitable for better scaling and results. For corporates having offices across the globe, this becomes even more necessary for better resource utilization. Using cloud storage would enable them to access any data from any part of the world while also allowing them to use unused resources from any of their offices. Thus, it also becomes very cost effective.

There are other benefits of cloud storage as well apart from the possibility of remote access/store of data. In case of natural disaster or power cut in some region, the data does not get lost because of replication policies. In fact, with modern techniques complete data can be recovered very fast.

High-scalability can also be ensured as nodes (read more storage space) can be easily added to the existing structure on demand. Bandwidth usage can also be optimized as sharing the data concerns sharing only access links. The complete data need not be transferred and only the relevant information can be accessed using the shared link.

Thus, it is safe to say that cloud storage, with such varied benefits, is definitely a very exciting technology. To make proper selection of cloud storage architecture, in this work, a logical flow is maintained and the end section lists a general algorithm that will assist in making an appropriate choice. This section is followed by the famous CAP (Consistency-Availability-Partition Tolerance) theorem, which states the tradeoff needed amongst these three parameters for any storage architecture. Then the report explores some famous storage architectures, implemented by major corporates, and a few open-source architectures which might be replicated by other service providers as well

The next section describes about various types of operation that an application might require to do on any storage architecture. This classification helps later in selection of storage architecture.

The penultimate section considers the various classifications described in an earlier section and determines which architecture would be suitable for the particular type of operation. In the end, a few real-life applications are considered, classified into various types of operations and then a storage architecture is determined.

2. CAP THEOREM (CONSISTENCY-AVAILABILITY-PARTITION TOLERANCE)

Any service is expected to give the user a continuous and accurate response. So as the applications move to the cloud, they need to ensure the user experience is not deteriorated. But as failure of servers is a normal case within cloud storage, the experience is not guaranteed. An interesting tradeoff among the desired properties of a storage system was highlighted by Eric Brewer.

Eric Brewer of UC Berkley, in his keynote speech [2] at the ACM Symposium on the Principles of Distributed Computing (PODC), 2000 gave the idea of the Cap Theorem. It was later in 2002, Gilbert and Lynch of MIT, proved his claims to be correct and CAP theorem came into existence. In this Theorem, he states the relation between three of the most desirable properties, namely, Consistency, Availability and Partition-Tolerance. In any network shared-data system, out of these three properties only two can be possible. Having all three simultaneously is not feasible [3].

Consistency: Consistency requires the change in data to be reflected across all data nodes immediately. The change must be visible atomically. At any point in time, different storage nodes should not be possessing different versions of data. Any discrepancy could lead to inaccurate information delivered to the user application.

Availability: The data should always be accessible. At every point of time, the request by the application should be responded with success or failure. The key test of availability occurs during its most busy period. As the server/node is most loaded during its busy time, the probability of failure is maximum.

Partition-Tolerance: Partition-Tolerance means that the system should continue its processes despite failure of nodes. Even if a partition is created between network sources, or one or more nodes fail, or proper communication is not possible between one or more subsystems, the process going on should not be interrupted [4]. The processing should continue in both the sub-groups in case of a partition.

Now according to the theorem, only two of these properties can be accommodated at one time (Figure 1).

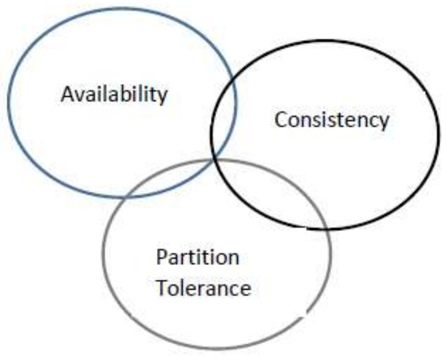

Fig 1: CAP theorem

Many applications require data storage in database form (could be for recommendation systems, frequent querying may be required etc). The two famous models of database storage, ACID (Atomicity –Consistency- Isolation-Durability) (relational databases) and BASE (Basically Available, soft State, eventual consistency) (NOSQL). These models also agree with the CAP theorem, in the sense that the selected properties are such that availability and consistency do not occur at the same time [3].

Most architectures these days, go for eventual consistency or eventual availability, in case of distributed systems. These terms mean that the system will be consistent or available but not at that instant. Although it is taken care using the architectures that the process is not interrupted.

3. MAJOR ARCHITECTURES OF CLOUD STORAGE

As discussed earlier, storage of data in cloud computing is very important. The latency in accessing and storing data, considerably impact the performance of the system. In other words, proper storage architecture can improve the performance of application many folds.

There are many different types of storage architectures currently being deployed for various types of applications. Some of these are used for internal purposes of the organization whereas some are used in the form of IaaS (Infrastructure as a Service). Amazon S3, Google Drive Storage, Dropbox etc are all examples of IaaS. Since there is a commercial value associated with their architectures, detailed description about them is not available.

However, a considerable amount of information is available about Dynamo/GFS, storage architectures for some internal purposes of Amazon and Google respectively. New technologies such as RAMCloud, Megastore are also elaborated in this section. Besides a few open-source architectures such as HDFS and OpenStack are discussed as well. While dynamo is primarily used for storing databases and other form of data having small size (~1 MB), GFS works well with even broader types of data (though it is optimized for storage of large files). Megastore is designed for interactive applications and RAMCloud increases access rate significantly in trade-off to availability. This section gives a detailed description about each of these storage paradigms.

The purpose to discuss these technologies, at logical-hardware-access level, is that the mechanism of implementation is very important to understand, for selection of architecture for any type of application. With a better understanding of these technologies, determining the storage architectures for some different type of classification (which are not included here) will also be easier.

After describing these technologies, next will be classification of different types of operation, followed by selection of one of these architectures for each operation.

Table 1 describes the features (from CAP) provided by various storage architectures. It is important to choose proper structure, based on the requirements of the application to be deployed. This table in brief describes about the priorities of each storage architecture.

Table 1: CAP theorem among architectures

Name of Architecture	Consistency	Availability	Partition-Tolerance
Dynamo		✓	✓
GFS		✓	✓
RAMCloud	✓		✓
Megastore		✓	✓

4. DYNAMO/OPENSTACK

Dynamo was designed by Amazon for having an always ‘write’ structure to support its e-commerce operation: www.amazon.com . Owing to the CAP theorem, dynamo sacrifices consistency across various back-ups. Object versioning is the collision resolution technique used in Dynamo [5].

Since Amazon is an ever expanding and growing firm, with data for each user also increasing with each passing day, the storage architecture deployed should be highly scalable. Reliability is another factor considered very seriously in dynamo’s structure. Failure is treated as a norm in dynamo’s architecture instead of a special case. Dynamo uses a single key structure, which meets requirements of many applications. Dynamodb is a NoSQL database service providing very high scalability built on dynamo structure [6]. The key target is applications with very small object sizes (~ 1MB) [5].

Consistent hashing is used for replication and partition of data [7] while eventual consistency is attained through object versioning [8]. Gossip protocol propagates information when a failure is detected and membership protocol is changed. Dynamo also uses a carefully designed mix of stateless (A service that compiles results from other services) and stateful (These type of services generate their own results using some well-defined logic) services. Security issues, such as authentication and authorization are not considered here as the use of storage structure is expected for internal purposes.

Figure 2 below gives a brief view into how the amazon platform functions. Whenever a request is made by the client, many stateful services combine to generate address of the stored data. This address is passed on to another server which directs the request to the appropriate date store. The stateless services in the data warehouse then redirect the information to the desired storage service (S3 or Dynamo) [5].

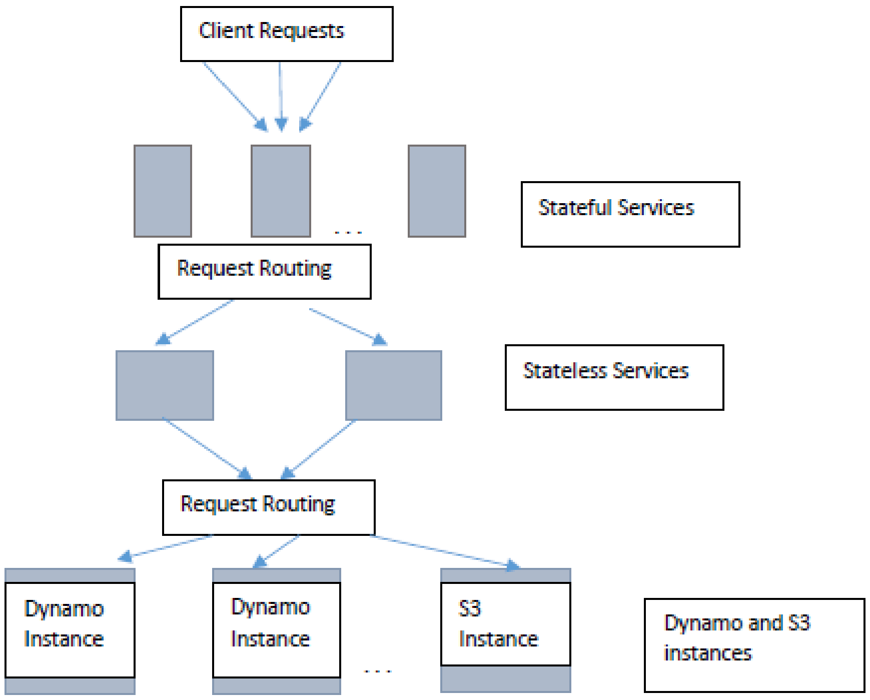

Fig 2: Amazon's platform

Applications have a major say in conflict resolution. They can choose to merge all records or keep the latest version depending on the need of application. No write is ever ignored, whatever internal failures might occur.

Data Access: get (key) and put (key) are used to access or insert values corresponding to key.

Partition Algorithm: A modified version of consistent hashing is used. Each node is placed multiple times in a logical ring (known as virtual nodes). The benefits being in case of failure, work distributed across the system fairly. Similarly on introduction of new node, data transfer almost equally from each node. Figure 3 shows a logical ring with each node replicated thrice. [5]

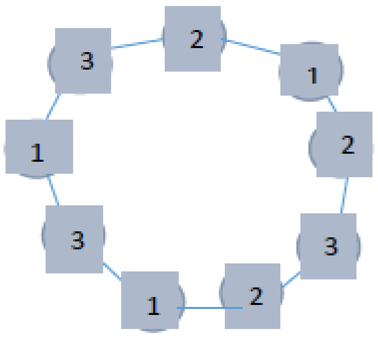

Fig 3: The Logical Ring for Replication

Replication: Each key k is assigned a main coordinator and key is replicated a fixed (N) number of times (based on service-level agreement) in a clockwise manner across the logical ring. Due to presence of virtual nodes, special care is taken to ensure that N different replicas are created.

Data Versioning: Data versioning is done using vectors [8]. A vector is maintained for each record and is updated whenever a write is done. If the first write is done by a node Sx, the vector is written to be D1[(Sx, 1)]. If the node Sx is the coordinator, and the next write is also done by Sx, it is updated to be D2[(Sx, 2)]. If for some reason during the next update, Sx was down, and Sy makes an update, vector clock is updated to be D3[(Sx, 2), (Sy, 1)]. If the vector size increases beyond a pre-specified limit, the vector is truncated based on time-stamps.

Failure Handling and Replica synchronization: Dynamo uses merkle trees [9], with hashes as leaves of keys [5]. This is done to reduce the amount of data transfer when checking for replica consistency. The structure can be tuned to always accept reads, or always accept writes.

The above structure is the basic layout and many different logical modifications can be implemented to tune the architecture as per requirement. In depth detail is accessible in [5]. OpenStack is an open-source cloud structure, with one of its instance swift having a structure very similar to dynamo, but based on BLOB and container-wise storage.

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

5. GFS (GOOGLE FILE SYSTEMS)/HDFS (HADOOP DISTRIBUTED FILE SYSTEMS)

GFS was implemented to match the continuous scaling needs at Google. This structure was designed to meet the storage needs of higher-size files. Another assumption made during design was that most of the data was to be appended rather than overwritten. GFS trades consistency for easier implementation of the system. Atomic append is also supported to reduce synchronization problems [10].

Here again, similar to dynamo, failure is treated as a normal case instead of an exception. Also optimization in the structure has been carried out with respect to larger file sizes, though smaller file sizes are supported as well. The expected workload is large sequential read/write. Several read/write maybe taking place simultaneously, even on the same file.

The basic architecture compromises of a single master and several chunk-servers. The servers can be accesses by multiple users at a time. Files are split into several chunks of pre-determined size. The size should be selected optimally, a very small size increases the storage overhead of metadata. A 64-bit handle is used to identify the chunk servers. Reliability is ensured by replicating data across multiple chunk-servers depending on SLA's.

Master server does not store data, it keeps track of status of chunk-servers and data on chunkservers using meta-data. The meta-data has information about garbage collection, lease, access controls etc. The state is updated using heartbeat communication (to and fro pings) with each chunk-server.

Whenever a client needs to read or write data, it approaches the master to know which chunk server should be contacted. For some time this information is cached in the client, as generally there are subsequent operations on the same chunk-server. This is done so as to reduce the workload of master and improving latency. Figure 4 explains the complete process of data access/write.

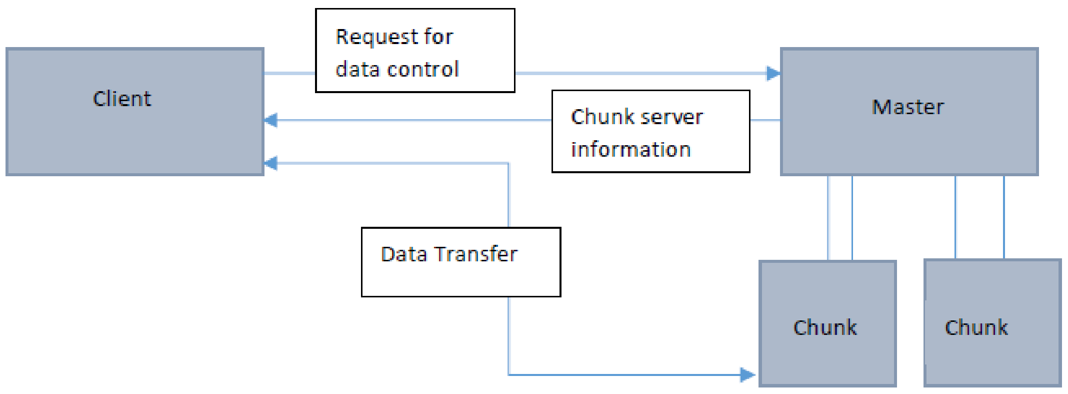

Fig 4: GFS

Data Access: Usual file operations such as create, open, read, close, write and delete are supported. Additional features such as snapshot (creating a local image) and record append (multiple atomic simultaneous write) were also introduced. Record append is useful in merging different works with very less synchronization overhead.

Replica Placement: The number of replicas are decided by service level agreements. The replica chunk servers are stored across different racks, to ensure that even when a complete rack is offline or unavailable due to some sort of damage, the operations on data are still possible. The placement of replicas across racks is determined by master and priority is given to ensure maximize availability and reliability. The master logs and checkpoints are replicated on many other machines. Thus even if the master goes down, the system stays available.

In case of a complete failure or permanent damage, as soon as the number of replicas reach a number less than desired, the master clones a chunk-server in high priority to ensure data loss or unavailability does not occur.

Whenever a request is made by one of the clients, the master grants a chunk lease to one of the replicas called primary [10]. All the writes/reads are managed by the primary until the lease expires (a timeout).
While the control flow takes place via the primary, the data flow is more carefully chosen. Data is pushed linearly across replicas [10]. Each machine forwards the data 'closest' to it. This is done so as to maximize the bandwidth usage. As soon as the chunk server receives data, it also starts forwarding it to other replicas (as shown in Figure 5).

Now Google uses a new system, colossus, details about which are not publically available. HDFS is an open source storage structure with similar topology.

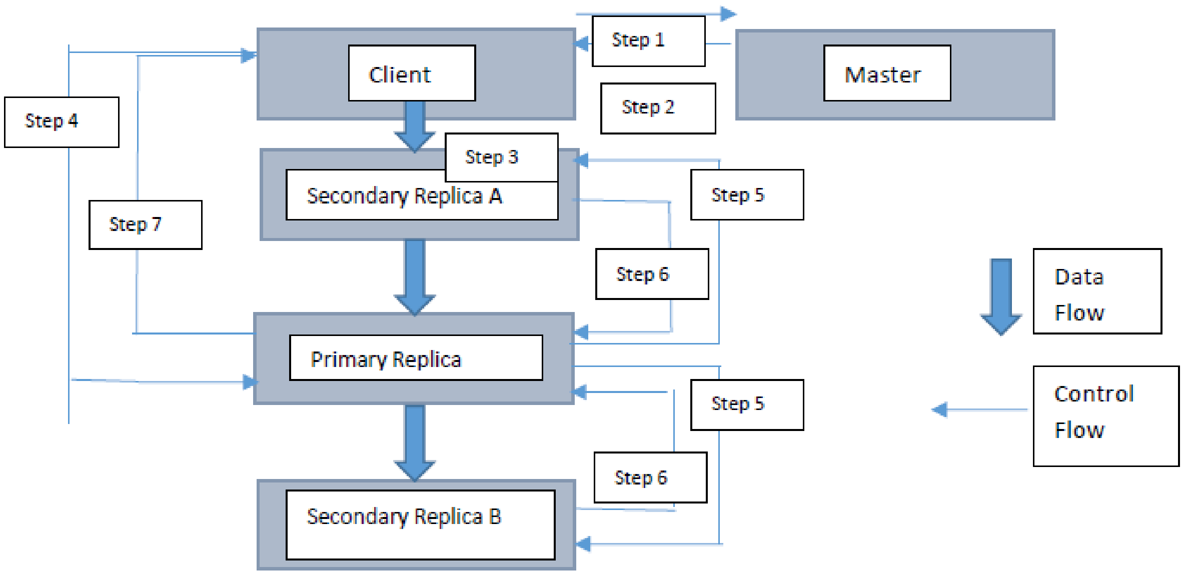

Fig 5: Control and Data Flow [10]

6. RAMCloud

Traditionally, DRAM was used only for caching. RAMCloud uses an entirely different structure. Here the primary storage is the DRAM and backup is done on disks. The main advantage of using a RAMCloud would be much lower latency (100x-1000x) and way better throughput [11]. One of the key feature it must support is automatic scalability to thousands of storage servers.

There is a possibility of achieving 5-10micro sec latency, which cannot be supported by existing switches and network. Still a major improvement in access speed is observed using RAMCloud. Though one drawback is in case of a system crash, data becomes unavailable for some time (1-2 sec) as data is to be recovered from backup disks [12].

The hardware for RAMCloud is made of hundreds and thousands of servers in a datacenter (each with capacity of 24-64 GB) divided into multiple clusters. Each cluster has a master and a backup. The backup stores replicas of other masters on disks or flash drives. Also each cluster has a dedicated coordinator to store location of objects and network addresses. The client can cache the memory address provided by coordinator, and then does not need to interact with it. In case of failures or crash, the client requests for new location of the data, and caches it.
Recovery of data is fastened using a log-structured storage. At every write request, data is logged and stored into a buffer. When the buffer is full, data is transferred to the back-up files. In case of crash, the only risk is to the data stored in the buffer. Though alternatives like separate power back-up etc can be used to counter this. For recovery, several masters are allocated. Each recovery master generates the hash table from log-structured data which is later merged. The key to fast recovery is utilizing the scale of the RAMCloud cluster.

RAMCloud is not the most efficient structure for storing files with large sizes, disks store these in a much better way [11]. A key-value type structure is best supported by RAMCloud [12]. The data could be a collection of innumerable tables containing innumerable objects (~ 1 MB)

Consistency is given the priority over availability. When a master fails, it stops servicing requests. If a coordinator fails, special care is taken that during the process only one of the coordinator stays active i.e. just after recovery as well both standby coordinator and the main one are not active at the same time.

7. MEGASTORE

This architecture is best suitable for services requiring some form of interaction with its clients. Megastore was designed to merge the scalability of NoSQL data store and convenience of RDBMS [13]. While NoSQL is high scalable, the relational databases provide convenience in building applications with its unique set of features. In brief it provides fully serial-izable ACID semantics across regions at very low latencies, which would be the perfect way to support interactive applications [13].

Partition of data is carried out, and each partition is replicated separately. Each partition independently provides the ACID semantics. A tolerable latency limit is set by the user, and all MySQL features that can scale within the limit are provided.

The data is to be replicated over wide regions for latency and availability constraints to be satisfied. Paxos algorithm is used, a consensus based algorithm [14]. Simply, a majority of replicas should be active during any write operation, which can be later propagated to other servers.

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

Reads, usually can be carried out on any replica, with no need to have inter- replica communication. A coordinator service is designed, having servers at all data centers. The coordinator keeps track if the requested data is fully updated or not in that particular data center.

A modified master ideology is used for write operations. Instead of masters, 'leaders' are chosen for each log-position. The writes at each log position is managed by a replica which is expected to coordinate with the leader of the previous log-position (to validate if previous write was successful or not) [13]. Paxos is run for each log-position separately. Generally the leader assigns the closest replica to be the 'master' for any write operation.

Read-only replicas are also made, for cheaper storage. They do not take part in consensus but store the snapshot of the data for access. There is also a concept of witness replica that take part in consensus, stores write-log but not indexes or entity. They are effectively used as tie-breakers.

In megastore availability is given preference over consistency. But with latency time usually comparatively lesser than access interval, the data is transferred across replicas efficiently and appears to be consistent.

8. CLASSIFICATION OF DIFFERENT TYPES OF OPERATIONS ON STORAGE

This sections broadly lists the various operations that may be carried out on our storage structure. Every application generally requires a combination of these operations, determining which is very important for designing/ selecting an efficient storage architecture.

Small Read/Write

In these type of operations, the data size is generally very small (maybe a few MB's). Applications requiring storing user data for later use is an example. Google stores data of each user- browsing history, locations visited, interests etc. This data is later used for recommendations and targeted advertisements. All of these writes can classify as small writes. Most of the databases stored and maintained would also qualify within this category.

Large Read/Write

Any data that exceeds the above mentioned size-range can be classified within this category. Backups, Logs, or any other general form of data will require long sequential read/write operations. Such data is very rarely overwritten.

Guaranteed Write

Some applications may require the data to be always written, irrespective of failures or consistency. E-commerce websites like amazon, flipkart, ebay etc would want to store information about all the searches carried out by the user, for better user experience. These type of applications require that data is stored at least in one of the replica, even though all other systems are down. The system may be inconsistent for some duration but that does not affect the application performance by a huge margin.

Fast Access

While every application needs fast access, some might need to thousands of queries to be replied with very low latency (Traditional databases like MySQL cannot scale). This form of access might be very useful where applications need to process data-intensive graph like algorithms (parallel processing not possible, current request depends on the previous one).

Highly Consistent

Data availability is not a big concern. More important is the accuracy of the data delivered. Applications like ticketing and hotel booking would require correct data given to the user at any time. Display of wrong quantity of available tickets could lead to serious issues. Even during peak hours, latency is acceptable but lack of consistency is not.

Highly Interactive

Applications like email, maps, collaborative applications like online docs etc need high user interaction. Response in such cases must be quick, consistent and always available. But this does not fit in well with the CAP theorem. A special type of structure must be developed, which is always available and appears consistent.

9. SELECTION- OF STORAGE ARCHITECTURE ON BASIS OF PERFORMANCE PARAMETERS

Here listed are some possible storage structures for each type of operation. Corresponding to each type of usage, an architecture has been suggested from amongst the ones described already. Though other alternatives might also be available for storage.

Small Read/Write

Database oriented architectures like Dynamo would be perfect for such applications. They are highly decentralized, thus proper workload distribution. Data access is possible simply using the key. For randomized data (not key based), Google File Systems can be used but no special optimization is carried out for small writes. For randomized data, RAMCloud will serve better purpose with its superior speed. Availability maybe poorer than other architectures (for RAMCloud), but might still be good enough for randomized data (as recovery rate is fast).

Large Read/Write

Architecture similar to Google File System would be a fitting choice in such a case. The whole system has been

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

designed specially assuming large sequential read and write. The complete concept of caching addresses of memory has been introduced for longer operations on the memory.

Guaranteed Write

RAMCloud is not the good choice for such requirement as in case of the main memory crash (DRAM crash), data will not be written. Dynamo fits as a perfect choice here, as even on failure or crash of a node, the node adjacent to it in the logical ring is expected to store the data temporarily. Amazon itself developed dynamo keeping this particular feature as the priority for its e-commerce platform. Megastore can also be used, for this purpose, but an unnecessary latency of managing relations might be added in such a case (If the stored data is to be NoSQL type).

Fast Access

RAMCloud is the ultimate choice in this case. In fact with better technology available for network switches and routers, the current latency can also be improved by a considerable margin. DRAM supports 10-1000x faster access than conventional disks. For huge files, still Google file system has to be used as data location will have to be accessed each time, irrespective of the data being sequential or not.

Highly Consistent

RAMCloud explicitly ensures consistency over availability. With access to data only through DRAM (data cannot be accessed/written from/to backup disks). This more or less ensures that data available will always be consistent. Megastore can also be a decent choice. Firstly because it provides better availability and secondly as generally the data is stored/accessed from the nearby replica, the data is usually consistent. Even if that is not the case, coordinator ensures consistent data in trade off to latency due to access from the next nearer replica.

Highly Interactive

Megastore has been developed by Google especially for this purpose. Usually interactive services would need to store data using relational database features. With storage and access mechanism arranged so that nearest replica to be used, the latency will be low – very important for interactive services (user cannot be left waiting for long). Also ability to store relational data is provided individually in each data center is a major asset for these types of application.

Table 2 summarizes the complete discussion described in the section.

Table 2: Recommended Storage Architecture Based on Type of Operation.

Type of Operation	Recommended Storage Architecture
Small Read/Write	Dynamo
Large Read/Write	GFS
Guaranteed Write	Dynamo / Megastore
Fast Access	RAMCloud
Highly Consistent	RAMCloud
Highly Interactive	Megastore

10. EXAMPLES OF STORAGE ARCHITECTURE SELECTION ON BASIS OF APPLICATION

Online Shopping/ E-commerce

Dynamo is developed specifically for e-commerce platform. It generally requires small writes (user preferences to be saved). Also key based reads will be needed for any item searched. Dynamo is based on an always-write philosophy, much needed for recommendation systems of the platform.

Ticketing

High level of consistency is expected in the data here. Though availability can be compromised a little, but most of the times, user needs to be able to access the data. RAMCloud can be deployed for such services as it provides very high consistency. Though structures like Megastore can be useful as well for services offering different kinds of packages and combinations (All nearby locations, tourist places can be stored in MySQL local tables).

Email/Blog/Map

All of these applications are interactive applications. A blog may be contributed by many writers whereas a map can be explored by the user in any way he wants. Megastore ensures quick response to the user irrespective of replication is complete or not. Within collaborative works, the contributing user sees the update immediately while other users may experience some acceptable amount of latency.

Netflix/Youtube

Netflix traditionally uses a SQL structure implemented on Google File system like architecture. One of the columns in each entry contains the video stored as an object. This makes it easier to search videos and also suggest related content (using tags stored in one of the columns). Also the access rate of old videos is very low, thus Google File System's assumption of very low over-write/read fits well. For popular videos, to manage the excess workload a special system known as content delivery network is used.

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

NoSQL Data for Data Mining

RAMCloud can be a good choice here, as data to be stored generally have no common link. Also when looking for some patterns, data will need to be accessed again and again. Faster access of RAMCloud will be an added bonus.

Data Archival

This kind of data is generally continuous and huge in size. Most of this data is rarely accessed and rarely overwritten. Storage architecture like Google File System will be perfect for such roles. The disk storage here is specially optimized for large files.

Table 3 below categorizes the variety of operations expected for each application and the recommended storage architecture for each application.

Table 3: Recommended Storage Architectures for Different Applications

Application	**Types of operations expected**	**Recommended Storage Architecture**
Online-Shopping/E-commerce	Small read/write	Dynamo
Ticketing	High Consistency	RAMCloud
Email/Blog/Map	High Interactivity	Megastore
Netflix/ Youtube	Large Files with inconsistent number of accesses	GFS
NoSQL for data mining	Fast and Random Access	RAMCloud
Data Archival	Large read/write	GFS

One simple methodology is to be used for determining the architecture. Firstly, the application needs to be analyzed and the type of operations that will be performed on the memory must be decided, as done in the previous section. Then prioritization should be carried out amongst the selected operations. Based on the few high priority operations, the storage architecture can be decided but also that the other operations are within acceptable latencies should be ensured. Post this, initial server capacities need to be decided based on the expected workload.

Thus, using the simple procedure described above, proper selection can be made and the best possible performance of the service can thus be obtained.

11. CONCLUSION

Usage of cloud has been growing rapidly due to its scalability, availability and backup benefits. It is inevitable to move applications on cloud for service providers as the application grows and thus it was important to understand what different type of storage structures are possible.

All service providers try to ensure that the selected architecture is the best suited for their application. As seen above, many corporates even develop their own architectures, so as to make the optimum use of their hardware resources and they are able to deliver the best possible performance. To select/design a storage structure for any application, different factors must be taken into consideration and then according to their priority, a storage architecture has to be finalized.
Storage is an inseparable part of any application and also makes a deep impact on performance of final product. Hence proper analyzing and research must be done before concluding to a particular architecture.

REFERENCES

[1] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich and F. Galan, "The Reservoir model and architecture for open federated cloud computing", *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1-4:11, 2009.

[2] E. BREWER, *PODC-keynote*.

[3] E. Brewer, "CAP twelve years later: How the "rules" have changed", *Computer*, vol. 45, no. 2, pp. 23-29, 2012.

[4] M. Stonebraker, ""Errors in Database Systems, Eventual Consistency, and the CAP Theorem", *Communications of the ACM*, 2010.

[5] *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" , ACM SIGOPS symposium on Operating systems principles (SOSP &#39;07), vol. 21, pp. 205-220, 2007.*

[6] "AWS Documentation", *Amazon Web Services, Inc.*, [Online]. Available: http://aws.amazon.com/documentation/.

[7] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. 1997. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on theory of Computing* (El Paso, Texas, United States, May 04 - 06, 1997). STOC '97. ACM Press, New York, NY, 654-663.

[8] Lamport, L. Time, clocks, and the ordering of events in a distributed system. ACM Communications, 21(7), pp. 558- 565, 1978.

https://doi.org/10.15623/ijret.2018.0710007 Received: 06-08-2018, Accepted: 17-09-2018, Published: 08-10-2018

[9] Merkle, R. A digital signature based on a conventional encryption function. Proceedings of CRYPTO, pages 369– 378. Springer-Verlag, 1988.

[10] S. Ghemawat, H. Gobioff and S. Leung, &quot;The Google file system&quot;, ACM SIGOPS Operating Systems Review, vol. 37, no. 5, p. 29, 2003.

[11] J. Ousterhout, G. Parulkar, M. Rosenblum, S. Rumble, E. Stratmann, R. Stutsman, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan and D. Ongaro, ”The case for RAMCloud”, Communications of the ACM, vol. 54, no. 7, p. 121, 2011.

[12] D. Ongaro, S. Rumble, R. Stutsman, J. Ousterhout and M. Rosenblum, “Fast crash recovery in RAMCloud”, SOSP ’11, pp. 29-41, 2011.

[13] J. Baker, C. Bond, J. Corbett, J. Furman, A. Khorlin, J. Larson, J. L´eon, Y. Li, A. Lloyd and V. Yushprakh, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services", 2016.

[14] L. Lamport, D. Malkhi, and L. Zhou. Vertical paxos and primary-backup replication. Technical Report MSR-TR 2009-63, Microsoft Research, 2009.

[15] "Virtualization in Cloud Computing", J Inform Tech Soft Engg, vol. 04, no. 02, 2014.