Performance of Scalable Data Stores in Cloud

Pankaj Deep Kaur, Gitanjali Sharma

Abstract*— Cloud computing has pervasively transformed the way applications utilized underlying infrastructure like systems and software. System designers are in fast track pursuit of deploying applications/services over cloud to benefit from its elastic, scalable and pay-as-you-go model. Owing to the fact that many applications on cloud are extensively data driven, data management systems, hosting these applications, embody a vital component in cloud software store. However, maintaining performance of database read/write operations under fluctuating workloads, both regionally and globally, is quite challenging. In this context, distributed scalable data stores in cloud have promised high performance and reliable services through rapid partitioning, replication, elasticity and automated manager for self-management. Thus, the success of cloud computing paradigm critically depends on scalable, elastic and automated DBMSs. This paper discusses state-of-art of techniques and technologies utilized for cloud databases. It presents concepts of partitioning, replication, elastic scalability and automatic manager for management. The paper also addresses challenges faced by DBMSs designers.

***Index Terms*— Amdahl's Law, Elasticity, Scalability.**

I. INTRODUCTION

Owing to technological proliferation, service providers are rapidly switching from computing infrastructure to cloud infrastructure, thereby giving rise to plethora of data. This has further led to an unsurpassed research and technological challenges in relation to database management systems. Unlike earlier scenarios, now an interruption in system has a far and wide global consequence making services unavailable to large number of users. Thus, experiences from past decades has led the DBMSs designers to design a scalable and reliable data store with high read/write performance even in the presence of fluctuating workloads.

Need for a scalable data store emerges as a result of fluctuating load characteristics over web-based applications. It ensures that by dynamically adding additional resources in accordance to the load characteristics, performance of system can be critically augmented. Systems can scale either vertically or horizontally by adding more resources to a single node or by adding more nodes respectively. Further, scalability of any system is associated with elemental algorithms and computations [3]. Specifically, if in an underlying algorithm a snippet α is intrinsically sequential then rest of 1- α is parallelizable [3] and hence, can gain from deployment of multiple processors. For such a system peak speedup or scalability of deploying N CPUs is confined as stated by Amdahl's law [1], [3].

$$Speedup = \frac{1}{\alpha + \frac{1-\alpha}{N}}$$

Revised Version Manuscript Received on June 29, 2015.

Pankaj Deep Kaur, Computer Science and Engineering, Guru Nanak Dev University- Regional Campus, Jalandhar, India.

Gitanjali Sharma, Computer Science and Engineering, Guru Nanak Dev University- Regional Campus, Jalandhar, India.

This implies that scalability is bounded by underlying algorithms [3] and can not necessarily be achieved by simply adding resources. Another factor closely related to scalability challenge is to device a mechanism for responding to unanticipated load variations. This mechanism is called elasticity. Scalability is static in nature i.e. it only guarantees that a system is capable of scaling from few hundreds to thousands of machines. However, elasticity is dynamic in nature and it permits operational systems to scale on demand. Another issue in guaranteeing database performance is automatic self-management, particularly in context of scalability, elasticity and load redistribution.

This paper briefly summarizes concepts of partitioning, replication, elastic scalability and automatic manager for management. It also addresses inherent challenges faced by ongoing researches in implementing these concepts over scalable cloud data stores.

Rest of the paper is structured as follows. Section II reviews partitioning strategies to support on demand scalability. Section III presents various replication approaches to guarantee all time availability of data. Section IV discusses how live data migration is achieved through elastic scalability. Section V presents the desired characteristics of automatic manager. Section VI reviews related work in the design space of scalable database systems. Finally, section VII concludes the paper with future suggestions.

II. DATABASE PARTITIONING

Partitioning helps in determining the location of data on the different geographically distributed servers in cloud. Thus, it is one of the important features in deciding the read, write or data storage performance and scalability [17] of the system. Partitioning of data can be achieved either through vertical partitioning or horizontal partitioning (also called sharding).

A. Vertical Partitioning

Vertical Partitioning is achieved by splitting columns (i.e. attributes) of database table into new subsets of columns such that there is a mapping between subset of columns and application functionality. As discussed in [21], it is essential to choose the appropriate table and columns in order to create a right partition, since the multi-table 'join' operations will now be carried within application code [21] and not over the relational schema. Thus, column family data stores are capable of providing both vertical partitioning and horizontal partitioning.

B. Horizontal Partitioning

Horizontal Partitioning or Sharding is achieved by splitting rows (i.e. tuples) of data base table into different tables with less number of tuples. Partitioning is achieved on the basis of shard keys which can either be directory-based, range-based or hash-based [21]. Based on the mapping of key and server mode, write requests are propagated to the appropriate server. Sharding based on later two keys is most commonly implemented by many major scalable data stores.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

Thus, two common techniques of sharding are: Range Partitioning or Consistent Hashing.

1) *Range Partitioning*

It is achieved by mapping data to multiple partitions spread over multiple servers on the basis on the range of shard keys [17]. This implies that one server is responsible for handling all read/write requests over a particular range of shard keys. However, it can result in hotspots or excessive load balancing problems. Moreover, this requires a routing server which will store the mapping of range to partitions as well as nodes and thus, help in routing requests to appropriate server. MongoDB [26], Cassandra [7], Hbase [18], and BerkleyDB [5], are among the few database that implement range partitioning.

2) *Consistent Hashing*

It uses hash key. The output range of hashing is a circular ring (i.e. largest value wraps to smallest value). The ring is split into number of ranges which is same as the number of nodes available. A random value from among the output range is assigned to each of the nodes thereby, determining their position in ring. Each data item is further mapped to a node in the ring by hashing its unique key to determine its position. Thus, each node is responsible for data region between itself and its predecessor [17]. Thus, there is no need to store mapping information. However, it is not efficient in range query processing as the consecutive keys are scattered across multiple nodes. As said in [17], this sharding is useful in effective dynamic resizing since, addition or removal of nodes will only require reassignment of neighboring areas and rest of the nodes remain unaffected. Voldemort [33], CouchDB [10], VoltDB [34], Clustrix [8], Riak [32] and Cassandra [7] use this type of sharding.

There are, data stores like Redis and Memcache, which implement no partitioning strategy at all and it is up to the client to come up with one. Amazon Simple DB provides its clients with a manual mechanism for partitioning but additional partitioning mechanism might be provided by service provider itself to achieve the desired throughput level as per service level Agreement (SLA) [17].

On the other hand, partitioning in Graph-oriented data stores [23] is complex to achieve owing to the highly mutable nature of graph data. Many graph partitioning mechanisms have been devised which try to achieve trade-off between two conflicting designs i.e. to store related graphs on same server to achieve effective query processing or to avoid storing too many nodes on the same server to prevent the need of load balancing. Graph data stores do not support stable shard keys and hence, do not implement partitioning as achieved by other data stores. For example: Neo4J [27] provides cache sharding [17] where as HypergraphDB [20] uses autonomous objects [17] to handle communication among graphs stored in neighboring per nodes.

Furthermore, there are NewSQL data stores like Google Spanner [9] and NuoDB [28] which implement slightly different mechanisms of partitioning as discussed briefly in [17].

III. DATABASE REPLICATION

Replication is the mechanism of storing multiple copies of same data over different servers in order to execute read/write requests by distributing queries over replicas. Mechanism adopted for replication influences performance of DBMSs. Besides being an important feature in determining scalability, it is also an essential feature in guaranteeing availability, fault-tolerance and affecting consistency level.

A. *Approaches to Replication*

According to Grolinger et al. [17], the main approaches to replication can be differentiated as: master-slave, multi-master or masterless replication.

1) *Master-Slave Replication*

In master-slave replication, one node is designated as master and rest as slaves. Only master is responsible for processing the write requests and propagating data to slaves. Thus, the direction of propagation is always from master to slaves. Some of the data stores implementing master-slaves replications are: Hbase [18], Redis [31], and BerkleyDB [5].

2) *Multi-Master and Masterless Replication*

In multi-master replication, any number of nodes can process the write requests and updates are then propagated to every other node. Thus, here propagation can be in any direction. Some of the data stores implementing multi-master replication are: Couchbase Server [11] and CouchDB [10]. Master-less replication is similar to multi-master approach except the fact that in former, all the nodes play same role in replication system [17]. Examples of data stores using master-less replication are: Cassandra [7], Voldemort [33] and Riak [32]. NewSQL data store achieve replications through Paxos state machine algorithm like in Google Spanner [9] or through transaction/session manager [17] like in VoltDB [34] and Clustrix [8], etc.

Further read and write performance or scalability of data store is affected by the choice of replication approach. Master-slave provides read scalability but not write. However, multi-master and master-less replication provide both read and write scalability owing to the fact that all nodes are allowed to handle both read and write requests.

B. *Update Processing Overheads*

Principle overhead in replication is the update processing required for remote as well as local propagation. As discussed by Kamal et al. [21], there are two main update processing operations: symmetric and asymmetric. Former processing requires a handsome amount of resources in remote replicas like CPU, I/O, etc and it achieves divergent consistency for non- deterministic db operations on local replicas and then binds the changes into write sets which are propagated to remote replicas as one single message. Depending on these overheads it is decided as to how the data is actually replicated [21] i.e. Full or Partial replication.

1) Full Replication

In Full Replication, each participating node has replicated copy of data and every remote replica has exactly same snapshot of local database. Thus, in the event of large number of update workloads, it poses a great deal of overhead as update processing is now required for multiple remote replicas.

2) Partial Replication

In Partial Replication, data is replicated to a group/subset of nodes instead of all participating nodes. Thus, update processing can be localized to a few replicas only. However, it also faces some challenges due to the dynamically changing

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

workload and application requirements as well as the complexity of determining the data item to be accused in replication. Two basic variants are: pure and hybrid partial replication [21].

a) Pure Partial Replication

In Pure Partial Replication, no participating node has full snapshot of local data base i.e. all nodes have partial copy of local database. It becomes difficult to predict to which replica has the desired data item to be accessed unless proper partitioning is done and workload is more and less static.

b) Hybrid Partial Replication

In Hybrid Partial Replication, some nodes have full copy of local data base while others have only a subset. Here the read requests can be localized for efficient processing and write requests can be distributed over different replicas. This can cause the overhead of creating hotspots and thus, the need for load balancing.

C. Replication Patterns

As discussed by Kamal et. al [21], web applications are basically deployed over a multi–tier cloud architecture where every single tier is responsible for handling the functionalities, coordinating with other tiers and providing desired services to the clients. Therefore, replicating a single tier is not an effective solution to achieve desired scalability. Besides read or write requests, there are compute intensive and data intensive operations. Former need more resources or scalability at application/logic tier and latter needs the same at data/persistence layer [21]. Moreover, in case of failures, interdependencies among tiers must not result in multiple execution of same workload at both application and database tier. Hence, considering above arguments, vertical and horizontal replication patterns are classified in [21].

1) Vertical Replication Pattern

This integrates one application and one database server into single replication unit which can be replicated vertically to achieve higher scalability. Here, replication logic is transparent to the replication unit, thereby enabling seamless working of the unit. However, it demands effective partitioning of application and its corresponding data to achieve expected scalability. Such systems are rarely used.

2) Horizontal Replication Pattern

This allows each tier to replicate independently and a replication awareness scheme [21] runs in between for coordinating the tiers. Thus, it provides flexibility to scale each tier independently but for effective performance the awareness mechanism is must, such systems are used almost everywhere.

D. Replication Architectures

Based on "Where" to implement replication logic, different replication architectures as presented by Kamal et al. [21] can be classified as shown in table I.

Table I. Replication architectures.

Replication Architecture	Implementation
Kernel-Based (White Box explication)	Logic implemented in database kernel
Centralised Middleware (Black Box Replication)	Logic implemented in middleware layer
Grey-Box Replication	Modified version of black-box replication. Explicitly presents concurrency control mechanisms by interfacing with middleware
Replicated Centralised Middleware Based	Backup of middleware is created
Distributed Middleware Based	Integrates every replica individually with middleware instance

IV. DATABASE ELASTICITY

Elasticity is the ability of the system to scale with load fluctuations by adding additional resources in the event of high workloads or by confining the tenants to less nodes during low workloads [3]. This is achieved dynamically in a live on-demand system without any disruption of services. Besides minimizing the operational cost, elasticity is also useful in live migration of database while having less impact on performance. Further, implementation of live database migration can be achieved over two major architectures of cloud data stores i.e. shared storage architecture and shared nothing architecture.

A. Shared Storage Architecture

Shared storage architecture stores the persistent database image in a network attached storage (NAS) [3] and is not migrated among the nodes. An iterative copy is designed for live database migration. Iterative copy lays emphasis on propagating only main memory state of a particular partition which consists of cached database state and transaction execution state like lock table, read/write collection of active or committed transactions. This minimizes the delay and downtime of tenant's window. It ensures transactional serializability during migration and transactional integrity in case of failures. HBase [18] and ElasTraS [12] use shared storage architecture.

B. Shared Nothing Architecture

In shared nothing architecture, each tenant has its own copy of database called partitions which are stored using locally attached storage. Unlike shared storage, here persistent database image is comparatively larger. Thus, it does not use iterative copy in order to avoid system downtime and service disruption, minimize data propagated among nodes and ensures safe propagation during failures. This approach does not count on replication thereby providing flexibility in choosing any location for migration. For example: Zephyr [3] uses this approach by introducing on demand pull and asynchronous push of data thereby allowing source and destination nodes to execute active and new transactions respectively.

V. AUTOMATIC MANAGER

Managing large database management systems presents remarkable challenges in system monitoring, operations and

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

management. Automatic manager is accountable for following:

- Monitoring system behavior.
- Tuning system performance.
- Elastic scaling.
- Load balancing on the basis of dynamic consumption patterns.
- Modelling system characteristics in order to predict the workload spikes.
- Taking effective control measures to deal with such spikes.

Further, in order to guarantee efficient multi-tenant performance and service level agreements (SLAs) [3], the manager must configure dynamic behavior and resource requirements of various tenants for elastic scaling. Also migration costs in context of decisions regarding where to migrate, when to migrate and which tenant to migrate must be predicted. Automatic manager comprises of two logical components: static and dynamic.

A. *Static Component*

Static component configures characteristics of tenants and their resource utilization to determine their placement and identify co-located tenants with supplementary resource requirements. This configuration presumes that once tenant characteristics are modeled and their placement is identified, system will retain its behavior and is thus called static component.

B. *Dynamic Component*

Dynamic component configures entire system's characteristics to govern appropriate moment for elastic load balancing. It guarantees minimum changes in tenant positioning and rebalances load through live database migration. Dynamic component identifies dynamic transformations in load and resource usage characteristics.

VI. Related Work

Peer-to-peer systems by [30] have been used lately to address the storage and distribution of data over network. These support flat namespaces. There were unstructured P2P [30] systems which usually broadcasted queries through network in order to hunt for as many peers possible which share same data. Examples include Freenet [2] and Gnutella [14]. Further evolutions led to structured P2P [30] systems which route queries to selective peers with required data using routing protocols like in Chord [2], Tapestry [2], CAN [2], etc. Later many systems evolved with the advances in efficient routing mechanisms like Oceanstore [29].

Distributed databases have been in use since decades to handle data storage and accesses through distributed transactions and query processing. Similar to distributed databases there are distributed file systems which act as object stores and use hierarchical namespaces. Some of these include Boxwood and Sinfonia. Further, distributed hash tables have also been used by certain projects like Chord [2], [15] Pastry [15], etc. Elastras presented by Agrawal et al. and Das et al. [3], [12] implements schema level partitioning [3] where different shards are independent of each other and hence, performance overhead owing to distributed transactions is omitted. However, here partitioning was static. Many common databases as evaluated by Das et al. [13], Lindsay et al. [22] provide iterative copy [3] guarantees which transfer main memory state of partition. Zephyr by Emore et al. [16] was designed to ensure serializability and transaction isolation [22] without relying on replication, thereby, reducing amount of data transferred among nodes.

Remuse described by Minhas et al. [25] on the other hand provides highly available data by using virtual machines and by preserving all ACID guarantees at the time of failure. Chimera by Minhas et al. [25] is another db architecture which operates on a hybrid platform of shared and shared nothing DBMSs. It can, thus, scale out elastically and balance load through data sharing. Further, discussions by Bhat et al. [6], Grolinger et al and Hu et al. [17, 19] present a detailed overview of other NoSQL and NewSQL databases on the basis of design decisions adopted.

VII. Conclusion

Database management systems on cloud form an integral component of cloud software stack. DBMSs designers are continuously facing various challenges to augment performance of cloud databases while minimizing operational cost of the system. The success of cloud computing is highly contingent on the effective design of scalable DBMSs. Hence, techniques discussed in this paper can be helpful in understanding and addressing the challenges for future advancements. Considering the previous and ongoing advancements in architectural design space helps us to conclude that upcoming scalable database projects must be capable of supporting automatic partitioning or replication in accordance with dynamic workloads. Also, the primary issue is to ensure rapid consistency with acceptable level of latency.

References

1. Amdahl's Law. http://en.wikipedia.org/wiki/Amdahl%27s_law [Online] Accessed 23 Nov 2014.
2. S. Androutsellis-Theotokis, A White Paper: A survey of peer-to-peer file sharingtechnologies. [Online] Accessed 23 Nov 2014. http://www.cs.ucr.edu/~michalis/COURSES/179-03/p2psurvey.pdf
3. D. Agrawal, A.E. Abbadi, S. Das and A.J. Elmore, Database scalability, elasticity and autonomy in the cloud [Extended Abstract] Technical report UCSB CS.
4. J. Baker, C. Bond, J.C. Corbett, J.J. Furman, A. Khorlin, J. Larson, J-M. Leon, A. Lloyd, V. Yuhprakh, (2011). Megastore: providing scalable, highly available storage for interactive services. Published under CIDR'11. Pages: 223-234.
5. BerkeleyDB.http://www.oracle.com/us/products/database/berkeley-db/overview/index.html Accessed 31 Jan 2015.
6. U. Bhat and S. Jadhav (2010) Moving towards non-relational databases. In IJCA (0975-8887) Vol. 1, No. 13.
7. Cassandra. http://cassandra.apache.org/ Accessed 31 Jan 2015.
8. Clustrix. http://www.clustrix.com/ Accessed 31 Jan 2015.
9. J.C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, JJ Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, D. Woodford (2012) Spanner: Google's globally-distributed database. Published in the proceedings of OSDI.
10. CouchDB. http://couchdb.apache.org/ Accessed 31 Jan 2015.
11. Couchbase Server. http://www.couchbase.com Accessed 31 Jan 2015.
12. S. Das, S. Agrawal, D. Agrawal, A. E. Abbadi, ElasTraS: an elastic, scalable, and self managing transactional database for the cloud. UCSB Computer Science technical report 2010-04
13. S. Das, S. Nishimura, D. Agrawal, A. E. Abbadi (2010) Live database migration for elasticity in a multitenant database for cloud platforms. Technical report, CS, UCSB(2010)

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.

14. H. C. Ding, S. Nutanong, R. Buyya. Peer-to-peer networks for content sharing. http://www.cloudbus.org/papers/P2PbasedContentSharing.pdf
15. S. El-Ansary, S. Haridi (2004). An overview of structured p2p overlay networks. http://eprints.sics.se/237/1/elansary-singlespaced.pdf
16. J. Emore, S. Das, D. Agrawal, A. E. Abbadi (2011) Zephyr: live migration in shared nothing databases for elastic cloud platforms. Published in ACM SIGMOD'11
17. K. Grolinger, W. A. Higashino, A. Tiwari, M. AM. Capretz (2013) Data management in cloud environments: NoSQL and NewSQL data stores. Journal of Cloud Computing: advances, systems and applications 2013, 2:22 doi: 10.1186/2192-113X-2-22
18. HBase. http://hbase.apache.org/ Accessed 28 Feb 2015.
19. H. Hu, Y. Wen, T-S. Chua, X. Li (2014) Towards scalable systems for Big Data analytics: a technology tutorial. In IEEE 2169-3536 Vol. 2, 2014 Pages 652-687.
20. HyperGraphDB. http://www.hypergraphdb.org/ Accessed 31 Jan 2015.
21. J.M.M. Kamal, M. Murshed (2014) Chapter 2 Distributed database management systems: architectural design choices for the cloud. Under Springer International Publishing- Mahmood (ed.), Cloud computing, Computer Communications and Networks
22. B.G. Lindsay (2008) Jim Gray at IBM. The transaction processing revolution. In SIGMOD Record Vol. 37, No. 2. Pages 38-40
23. T.S. Madhulatha (2012) Graph partitioning advance clustering technique. In IJCSES Vol. 3, No. 1. Pages 91-104.
24. Memcached. http://memcached.org/ Accessed 28 Feb 2015.
25. U. F. Minhas (2013) Scalable and highly available database systems in the cloud. PhD Thesis, University of Waterloo, Canada.
26. MongoDB. http://www.mongodb.org/ Accessed 31 Jan 2015.
27. Neo4J. http://www.neo4j.org Accessed 31 Dec 2014.
28. NuoDB. http://www.nuodb.com/ Accessed 28 Dec 2014.
29. Oceanstore. https://oceanstore.cs.berkeley.edu/ Accessed 12 Nov 2014.
30. Peer-to-peer.http://en.wikipedia.org/wiki/Peer-to-peer#cite_ref-19 Accessed 12 Nov 2014.
31. Redis. http://redis.io/ Accessed 23 Feb 2015.
32. Riak. http://basho.com/riak/ Accessed 28 Dec 2014.
33. Voldemort. http://www.project-voldemort.com/voldemort/Accessed 23 Feb 2015.
34. Volt DB. http://www.voltdb.com/ Accessed 12 Feb 2015.

Pankaj Deep Kaur, PHD, MIT gold medalist and UGC-JRF qualified, 32 publications in journals and conferences, specialization in cloud computing and big data, 35 publications in journals and conferences.

Gitanjali Sharma, B.Tech. in Computer Science and Engineering and currently pursuing M.Tech. in Computer Science and Engineering, Research area deals with cloud computing, big data and cloud databases, 2 publications in international conferences.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication Pvt. Ltd.