INTERNATIONAL JOURNAL OF COMPUTATIONAL MATHEMATICAL IDEAS
ISSN: 0974-8652 / www.sijcmi.weebly.com / VOL 5-NO1-PP 2000-2006 (2013)

Web-Based Database Applications In The Cloud

K.Rama Krishna Rao[*1], Dr.A.Srinivasa Rao[*2]

[1]*Associate professor, CSE Department, Ideal engg. College, Kakinada., India.*

*2P*rofessor, CSE Department, KL University, Guntur Dt., India

***Abstract*---**Cloud computing has seen tremendous growth, particularly for commercial web applications. The on-demand, pay-as-you-go model creates a flexible and cost-effective means to access compute resources. For these reasons, the scientific computing community has shown increasing interest in exploring cloud computing. However, the underlying implementation and performance of clouds are very different from those at traditional supercomputing centers. The Cloud has become a new vehicle for delivering resources such as computing and storage to customers on demand. Rather than being a new technology in itself, the cloud is a new business model wrapped around new technologies such as server Virtualization that take advantage of economies of scale and multi-tenancy to reduce the cost of using information technology resources. From one perspective, cloud computing is nothing new because it uses approaches, concepts, and best practices that have already been established. From another perspective, everything is new because cloud computing changes how we invent, develop, deploy, scale, update, maintains, and pay for applications and the infrastructure on which they run.

INTRODUCTION

With today's choices the possibilities of building applications inside the cloud are unlimited. Different services, possibly even from different providers, can be combined to achieve the same goal. Furthermore, by using virtualized machines, any kind of service can be created inside the cloud. Although there is no agreement on how to design applications inside the cloud, certain best practice recommendations exist. The simplest and probably fastest way to deploy an application inside the cloud is by use of a PaaS . The platform hides the complexity of the underlying infrastructure (e.g. load balancing, operating system, fault tolerance, firewalls) and provides all the services, typically through libraries, which are required to build anbnapplication (e.g., tools for building web-based user interfaces, storage, authentication, testing, etc.). As invoking services outside the platform provider is rather expensive with respect to latency, best practice is to use the services and tools from a single PaaS provider. Although PaaS offers many advantages, the biggest drawback is the incompatibility between platforms and the enforced limitations of the platform. In addition, by today, the PaaS software is proprietary and no open-source systems exist, thus preventing the use of this approach for private cloud installation.

Developers requiring more freedom and/or not willing to restrict themselves to one provider, may directly use IaaS offerings. These enable a more direct access to the underlying infrastructure. The canonical form of traditional web architectures Clients connect by means of a web browser through a firewall to a web server. The web server is responsible for rendering the web-sites and interacts with a (possibly distributed) file system and an application server. The application server hosts
and executes the application logic and interacts with a database, and possibly with the file system and other external systems. Although the general structure is similar for all web-based database applications, a huge variety of extensions and deployment strategies exists]. By means of virtualized machines all of those variations are also There are quantitative as well as qualitative benefits in maintaining quality assurance. The Quantitative benefits are reduced costs, greater efficiency, better performance, less unplanned work and fewer disputes. The Qualitative benefits are improved visibility and predictability, better control over contracted products, improved customer confidence, better quality, problems show up earlier and reduced risk.

Web-Based Database Applications in the Cloud

However, to profit from the elasticity of the cloud and the pay-as-you-go pricing model, not every architecture is equally applicable [Gar09]. For example, the quite popular monolithic (all-on-one server) architecture as epitomized by the LAMP stack (i.e., Linux, Apache, MySQL, PHP on one server) is not amenable to cloud applications. The architecture presents not only a single point of failure, but because it is limited to a single server instance, its

scalability in case that traffic exceeds the capacity of the machine is limited. The best practice deployment strategy for web-applications using
The application and the web server are deployed together on one VM, typically
by using a customized image. As discussed earlier, VMs normally loose all the data
if they crash. Thus, the web/application tier is typically designed stateless and uses other services to persist session data (e.g. a storage service or database service). The storage service (such as Amazon's S3) is typically used for large objects like images, or videos whereas a database is used for smaller records. The elasticity of the architecture is guaranteed by a load balancer which watches the utilization of the services, initiates new instances of the web/application server and balances the load. The load
balancer is either a service which might itself run on a VM, or a service offered by the cloud provider. In addition, the firewall is normally also offered as a service by the cloud provider. Regarding the deployment of database systems in the cloud, two camps of thought exist. The first camp favours installing a full transactional (clustered) database system (e.g., MySQL, Postgres, or Oracle) in the cloud, again by using a VM service. This approach provides the comfort of a traditional database management system and makes the solution more autonomous from the cloud provider.

On the downside, traditional database management systems are hard to scale and not designed to run on VMs [Aba09, Kos08]. That is, the query optimizer and the storage management assume that they exclusively own the hardware and that failures are rather rare. All this does not hold inside a cloud environment. Furthermore, traditional database systems are not designed to constantly adapt to the load, which not only increases the cost, but frequently requires manual interaction in order to be able to react to the load. The second camp prefers using a specially designed cloud storage or database service instead of a full transactional database. The underlying architecture of such a service is typically based on a key-value store designed for scalability and fault tolerance. This kind of service can either be self-deployed, typically using one of the existing open-source systems , or is offered by the cloud provider. The advantage of this approach lies in the system design for fault tolerance, scalability, and often self-management. If the service is operated by the cloud provider, it increases the outsourcing level. In this case, the cloud provider is responsible for monitoring and maintaining the storage/database service. On the downside, these services are often simpler than full transactional database systems offering neither ACID guarantees nor full SQL support. If the application requires more, it has to be built on top without much support available so far. Nevertheless, this approach is becoming more and more popular because of its simplicity and excellent scalability [1]..

Cloud Storage Systems

This section discusses cloud storage systems in more detail because of their fundamental role in building database application in the cloud.. Here, we use the name cloud and database service interchangeably, as none of the database services in the cloud really offers the same comfort as a full-blown database and, on the other hand, cloud
storage services are extended with more functionality (e.g., simple query languages) making them more than a simple storage service. Given the success of AWS and the recent offerings by Google, it is likely that the utility computing market and its offerings will evolve quickly. Nevertheless, we believe that the techniques and tradeoffs discussed in this and the subsequent chapters are fundamental and are going to continue to remain relevant.

Foundations of Cloud Storage Systems

The section explains the importance of the CAP theorem for developing cloud solutions before presenting some of the basic tools used to build cloud services.

The Importance of the CAP Theorem

To achieve high scalability at low cost, cloud services are typically highly distributed systems running on commodity hardware. Here scaling just requires adding a new offthe-
shelf server. Unfortunately, the CAP theorem states that it is not possible to achieve Consistency, Availability and tolerance against network Partitioning at the same time
[5]. In order to completely avoid network partitioning, or at least to make it extremely unlikely, single servers or servers on the same rack can be used. Both solutions do not scale and, hence, are not suited for cloud systems. Furthermore, these solutions also decrease the tolerance against other failures (e.g., power outages or
over-heating). Also, to use more reliable links between the networks does not eliminate the chance of partitioning, and increases the cost significantly. Thus, network partitions are unavoidable and either consistency or availability can be achieved. As a result, a cloud service needs to position itself somewhere in the design space between consistency and availability.

Consistency Guarantees: ACID vs. BASE

Strong consistency in the context of database systems is typically defined by means of the ACID properties of transactions [6]. ACID requires that for every transaction the following attributes hold: _ Atomicity: Either all of the tasks of a transaction are performed or none. _ Consistency: The data remains in a consistent state before the start of the transaction and after the transaction. _ Isolation: Concurrent transactions result in a serializable order. _ Durability: After reporting success, the modifications of the transaction will persist. If ACID is chosen for consistency, it emphasizes consistency while at the same time diminishing the importance of availability. Requiring ACID also implies that a pessimistic view is taken, where inconsistencies should be avoided at any price. As a consequence, to achieve ACID properties, complex protocols such as 2-phase-commit or consensus protocols like Paxos are required. On the other extreme, where availability is more important than consistency, BASE [Bre00] is proposed as the counter-part for ACID. BASE stands for: Basically Available, Soft state, Eventual consistent. Where ACID is pessimistic and forces consistency at the end of every operation, BASE is optimistic and accepts inconsistency. Eventual consistency only guarantees that updates will eventually become visible to all clients and that the changes persist if the system comes to a quiescent state [5]. In contrast to ACID, eventual consistency is easy to achieve and makes the system highly available. Between the two extremes BASE and ACID, a range of consistency models from the database community (e.g. the ISO isolation levels) [2] as well as from the distributed computing community [6] can be found. Most of the proposed models can lead to inconsistencies but at the same time lower the likelihood of those.

Techniques

To achieve fault tolerance and ease of maintenance, cloud services make heavy use of distributed algorithms such as Paxos and distributed hash-tables. This section presents the basic tool box to build highly reliable and scalable cloud services.

Cloud Storage Systems

basic techniques to compare different cloud services. However, the focus here is on distributed algorithms. Standard database techniques (e.g. 2-phase-commit, 3-phasecommit etc.) are assumed to be known.

Master-Slave/Multi-Master: The most fundamental question when designing a system is the decision for a master-slave or a multi-master architecture [2]. In the master-slave model one device or process has the control over a resource. Every change to the resource has to be approved by the master. The master is typically elected from a group of eligible devices/processes. In the multi-master model the control of the resource is not owned by a single process; instead, every process/device can modify the resource. A protocol is responsible for propagating the data modifications to the rest of the group and resolve possible conflicts.

Distributed hash-table (DHT): A distributed hash-table provides a decentralized lookup service [3]. Within a DHT the mappings from keys to values are distributed across nodes often including some redundancy to ensure fault tolerance. The key properties of a DHT are that the disruptions caused by node joins or leaves are minimized, typically by using consistent hashing [7], and that no node requires the complete information. DHT implementations normally differ in the hash method they apply (e.g. order preserving vs. random), the load-balancing mechanism and the routing to the final mapping [9]. The common use case for DHTs is to load-balance and route data across several nodes.

Quorums: To update replicas, a quorum protocol is often used. A quorum system has three parameters: a replication factor N, a read quorum R and a write quorum W. A read/write request is sent to all replicas N, and each replica is typically on a separate physical machine. The read quorum R (respectively the write quorum W) determines the number of replicas that must successfully participate in a read (write) operation. That is, to successfully read (write) a value, the value has to be read (written) by R (W) numbers of replicas. Setting $R + W > N$ ensures that always the latest update is read. In this model, the latency of read/write is dictated by the slowest of the read/write replicas. For this reason, R and W are normally set to be lower than the number of replicas. Furthermore, by setting R and W accordingly the system is balanced between read and write performance. The quorums also determine the availability and durability of the system. For example, a smallWincreases the chance of loosing data and if fewer nodes thanWor R are available, reads or writes are not possible anymore. For systems where availability and durability are more important than consistency, the concept of sloppy quorums was introduced.

Vector Clocks: A vector clock is a list (i.e, vector) of (client, counter) pairs created to capture causality between different versions of the same object [7].
Thus, a vector clock is associated with every version of every object. Each time a client updates an object, it increments its (client, counter) pair (e.g., the own logical clock) in the vector by one. One can determine whether two versions of an object are conflicting or have a causal ordering, by examining their vector clocks. Causality of two versions is given, if every counter for every client is higher or equal to the counter of every client of the other version. Else, a branch (i.e., conflict) exists. Vector clocks are typically used to detect conflicts of concurrent updates without requiring consistency control or a centralized service [7].

Paxos: Paxos is a consensus protocol for a network of unreliable processors [7]. At its core, Paxos requires a majority to vote on a current state - similar to
the quorums explained above. However, Paxos goes further and can ensure strong consistency as it is able to reject conflicting updates. Hence, Paxos is often applied in
multi-master architectures to ensure strong consistency - in contrast to simple quorum protocols, which are typically used in eventually consistent scenarios.

Gossiping protocols: Gossiping protocols, also referred to as epidemic protocols, are used to multi-cast information inside a system [8]. They work similar
to gossiping in social networks where a rumor (i.e., information) is spread from one person to another in an asynchronous fashion. Gossip protocols are especially suited for scenarios where maintaining an up-to-date view is expensive or impossible. **Merkle Trees:** A Merkle tree or hash tree is a summarizing data structure, where leaves are hashes of the data blocks (e.g., pages) [8]. Nodes further up in the tree are the hashes of their respective children. Hash trees allow to quickly identify if data
blocks have changed and allow further to locate the changed data. Thus, hash trees are typically used to determine if replicas diverge from each other.

Commercial Storage Services Amazon's Storage Services:

The most prominent storage service is Amazon's S3
. S3 is a simple key-value store. The system guarantees that data gets replicated across several data centers, allows key-range scans, but only offers eventual
consistency guarantees. Thus, the services only promise that updates will eventually become visible to all clients and that changes persist. More advanced concurrency control mechanisms such as transactions are not supported. Not much is known about the implementation of Amazon's S3. Section 3.2 gives more details about the API and cost infrastructure of S3 because we use it in parts of our experiments. Next to S3, Amazon offers the Elastic Block Store (EBS) [9]. In EBS, data is divided into storage volumes, and one volume can be mounted and accessed by exactly one EC2 instance at a time. In contrast to S3, EBS is only replicated within a single data center and provides session consistency [8]. Internally, Amazon uses another system called Dynamo [7]. Dynamo supports
high update rates for small objects and is therefore well-suited for storing shopping carts etc. The functionality is similar to S3 but does not support range scans. Dynamo
applies a multi-master architecture where every node is organized in a ring. Distributed hash tables are used to facilitate efficient look-ups and the replication and consistency protocol is based on quorums. The failure of nodes is detected by using gossiping and Merkle trees help to bring diverged replicas up-to-date. Dynamo applies sloppy quorums to achieve high availability for writes. The only consistency guarantee given by the system is eventual consistency, although the quorum configuration allows optimizing the typical behavior (e.g., read-your-write monotonicity). The architecture of Dynamo inspired many open-source projects such as Cassandra or Voldemort.

Google's Storage Service: Two Google-internal projects are known: BigTable [6] and Megastore [8]. The latter, Megastore, is most likely the system behind
Google's AppEngine storage service.
Google's BigTable [6] is a distributed storage system for structured data. Big- Table can be regarded as a sparse, distributed, persistent, multi-dimensional sorted
map. The map is indexed by a row key, a column key, and a timestamp. No schema is imposed and no higher query interface exists. BigTable uses a single-master architecture.
To reduce the load on the master, data is divided into so-called tablets and one tablet is exclusively handled by one slave (called tablet server). The master is responsible
for (re-)assigning the tablets to tablet servers, for monitoring, load-balancing, and certain maintenance tasks. Because BigTable clients do not rely on the master for tablet location information, and read/write request are handled by the tablet server, most clients never communicate with the master.

Chubby [Bur06], Google's distributed lock service, is used to elect the master, to determine the group-membership of servers, and to ensure one tablet server per tablet. As its heart, Chubby uses Paxos to achieve strong consistency among the Chubby servers. To store data persistently, BigTable relies on Google's File System (GFS) [3].
GFS is a distributed file system for large data files, that are a normally only appended and rarely overwritten. Thus, BigTable stores the data on GFS in an append-only mode and does not overwrite data. GFS only provides relaxed consistency but as BigTable guarantees a single tablet server per tablet, no concurrent writes can appear, and at row-level atomicity and monotonicity are achieved. BigTable and Chubby are designed as a single data center solution. To make BigTable available across data centers an additional replication protocol exists, providing eventual consistency guarantees [8]. Google's Megastore [8] is built on top of BigTable and allows to impose schemas and a simple query interface. Similar to SimpleDB, the query language is restricted and more advanced queries (e.g.,a join) are not possible. Furthermore, Megastore supports transactions with serializable guarantees inside an entity group.1 Entity groups are entirely user-defined and have no size restriction. Still, every transaction inside an entity group is executed serially. Thus, if the number of concurrent transactions is high, the entity group becomes a bottle-neck. Again, no consistency guarantees are made between entity groups.

Yahoo's Storage Service: Two systems are known: PNUTS and a scalabledata platform for small applications The first is similar to Google's Big-Table. PNUTS applies a similar data model and also splits data horizontally into tablets.

In contrast to BigTable, PNUTS is designed to be distributed across several data centers. Thus, PNUTS assigns tablets to several servers across data center boundaries. Every tablet server is the master for a set of records from the tablets. All updates to a record are redirected to the record master and are afterwards propagated to the other replicas using Yahoo's message broker (YMB). The mastership of a record can migrate between replicas depending on the usage and thus, increases the locality for writes. Furthermore, PNUTS offers an API which allows the implementation of different levels of consistency, such as eventual consistency or monotonicity.
Yahoo's second system consists of several smaller databases and provides 1Also, transactions are executed in a serial order; serializability can be violated because of the lazy updates to indexes .

Cloud Storage Systems

Nodes/machines are organized into so-called colos with one colos controller each. Nodes in one colos are located in the same data center, often even in the same rack. Nodes in a colo run MySQL and host one or more user databases. The colo controller is responsible for mapping the database to nodes. Furthermore, every user database is replicated inside the colo. The client only interacts with the colo controller and the controller forwards the request to the MySQL instance by using a read-oncewrite- all replication protocol. Thus, ACID guarantees can be provided. Additionally, user databases are replicated across colos using an asynchronous replication protocol for disaster recovery. As a consequence, major failures can lead to data loss. Another restriction imposed by the architecture is, that neither data nor transactions inside a database can span more than one machine, implying a scalability limit on the database size and usage.

Microsoft's Storage Service:

Microsoft offers two services: Azure Storage Service [Cal08] and SQL Azure Database [8]. Windows Azure storage consists of three sub-services: blob service, queue service, table service. The blob service is best compared to a key-value store for binary objects. The queue service provides a message service, similar to SQS, and also does not guarantee first in/first out (FIFO) behavior. The tablet service can be seen as an extension to the blob service. It allows to define tables and even supports a simple query language. Within Azure Storage Service data is replicated inside a single data center and monotonicity guarantees are provided per record but here exists no notion of transactions for several records. Little is known about the implementation, although the imposed data categorization and limitations look similar to the architecture of BigTable.
The second service offered by Microsoft is SQL Azure Database. This service is structured similar to Yahoo's platform for small applications but instead of running MySQL, Microsoft SQL Server is used [Sen08]. The service offers full transaction support, a simplified SQL-like query language (so far not all SQL Server features are exposed), but restricts the consistency to a smaller set of records (i.e., 1 or 10 GB).

Open-Source Storage Systems

This section provides an overview about existing open-source storage systems. The list is not exhaustive and many other systems such as Tokyo Cabinet, MongoDB

and Ringo exist. However, those systems were chosen as they are already more stable and/or provide some interesting features.

Cassandra: Cassandra was designed by Facebook to provide a scalable reverse index per user-mailbox. Cassandra tries to combine the flexible data model of BigTable with the decentralized administration and always-writable approach of Dynamo. To efficiently support the reverse index, Cassandra supports an additional threedimensional data structure which allows the user to store and query index like structures. For replication and load-balancing, Cassandra makes use of a quorum system and a DHT similar to Dynamo.

CouchDB: CouchDB is a JSON-based document database written in Erlang. CouchDB can do full text indexing of the stored documents and supports expressing views over the data in JavaScript. CouchDB uses peer-based asynchronous replication, which allows updating documents on any peer. When distributed edit conflicts occur, a deterministic method is used to decide on a winning revision. All other revisions are marked as conflicting. The winning revision participates in views and further provides the consistent view. However, every replica can still see the conflicting revisions and has the opportunity to resolve the conflict. The transaction mechanism of CouchDB can best be compared with snapshot isolation, where every transaction sees a consistent snapshot. But in contrast to the traditional snapshot isolation, conflicts do not result in aborts. Instead, the changes are accepted in different versions/branches which are marked as conflicting.

HBase: HBase is a column-oriented, distributed store modeled after Google's BigTable and is part of the of the Hadoop project, an open-source MapReduce framework [8]. Like BigTable, HBase also relies on a distributed file system and a locking service. The file system provided together with Hadoop is called HDFS and is similar to Google's FileSystem architecture. The locking service is called ZooKeeper [8] and makes use of the TCP connections to ensure consistency (in contrast to Chubby from Google which uses Paxos). However, the whole architecture of HBase, HDFS and ZooKeeper looks quite similar to Google's software stack.

Conclusion:

While cloud computing has proven itself useful for a wide range of e-Science applications, its utility for more tightlycoupled HPC applications has not been proven. In this paper we have quantitatively examined the performance of a set of benchmarks designed to represent a typical HPC workload run on Amazon EC2. Our data clearly shows a strong correlation between the percentage of time an application spends communicating, and its overall performance on EC2. The more communication, the worse the performance becomes. We were also able to see that the communication pattern of the application can have a significant impact on performance. It is concluded that transactional data management applications are not well suited for cloud deployment. The characteristics of the data and workloads of typical analytical data management applications are well-suited for cloud deployment.

References:

[1] S. Hazelhurst, "Scientific computing using virtual high-performance computing: a case study using the Amazon elastic computing cloud," in Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology. ACM, 2008, pp. 94–103.

[2] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: the montage example," in Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press, 2008, pp. 1–12.

[3] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," Cloud Computing and Applications, vol. 2008, 2008.

[4] K. Keahey, "Cloud Computing for Science," in Proceedings of the 21st International Conference on Scientific and Statistical Database Management. Springer-Verlag, 2009, p. 478.

[5] J. Napper and P. Bientinesi, "Can cloud computing reach the top500?" in Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop. ACM, 2009, pp. 17–20.

[6] Madden, S., DeWitt, D., And Stonebraker, M., Database parallelism choices greatly impact scalability. Database Columnblog.

[7] Stonebraker, M., Madden, S., Abadi, D., Harizopoulos,
S., Hachem, N., And Helland P., 2007, .The end of an architectural era . In VLDB., Vienna, Austria.

[8] Agrawal, R., Kiernan, J., Srikant, R., and Xu, Y., 2004, .Order preserving encryption for numeric data,. Proc. Of SIGMOD, pp. 563.574.