Databases in the Cloud

Bhuleskar, Ronald; Ko, Kenneth; Vanama, Srujan; Abualrob, Mohammed

Department of Computer Engineering, Santa Clara University, Santa Clara, CA

***Abstract*—Various aspects of Cloud Computing and Databases will be considered in order to evaluate and examine the need for DBaaS (Database-as-a-Service). We define Cloud Computing and related topics such as multitenancy and distributed computing algorithms, we examine, primarily, the Cloud-oriented services offered by various Cloud providers keeping our focus on Amazon; however, we provide overview of three large Cloud Provider's Cloud database solution by: Amazon, Google, and Microsoft. We classify few database services offered as SaaS and when you may use specialized Cloud services as PaaS and IaaS.**

***Index Terms*—Cloud Computing, Database, Database as a Service (DBaaS), Amazon Web Services (AWS)**

I. Introduction

Today most of the IT industry is transforming by moving applications, platforms, infrastructure and collaboration as well to Cloud.

A. Cloud Computing

Cloud computing is a service driven by very powerful benefits, whether it is hosted on a Private Cloud within an organization or provided by a third-party or even maintained as a hybrid of both. We will look at some of the benefits the Cloud provides for our business needs as provided by [1]:

- **Lower Infrastructure Costs** – No need to buy expensive equipment specifically for your organization as these can be shared among multiple clients using Cloud service.

- **Shifting Capital Expenditure to Operational Expenditures** – Capital expenditure is no longer needed; all you pay operating costs, allowing you to focus your resources on innovation.

- **Agility** – On-Demand service, faster setup and teardown of resources, add as you need and remove whenever you are done.

- **Dynamic Scalability** – No need to buy additional equipment to handle heavy load or spikes before hand. Cloud can efficiently handle these types of scenarios by allocating resources, as we need dynamically—pay as you go.

- **Easy Maintenance** – All the upgrades and interim fixes are deployed across the shared infrastructure at once.
- **Multi-platform Support** – Cloud Computing supports rich collection of platforms including applications, etc.

- **Easy Prototyping and Testing** – Easily allocate resources for your prototype and testing it and release the resources as soon as you are done.

- **Faster and Quicker Development** – Cloud provides most of the core services and setting up the required environment with the help of templates and modules, improving the pace of development cycle.

B. Databases in the Cloud

The database is a piece of software that organizes, stores and retrieves massive amount of data systematically and easily. A Relational database is a system that stores information in the tables.

The idea of databases in Cloud is the concept of providing on-demand access to database for storing or retrieving application data. Cloud database usage patterns are evolving, and business adoption of these technologies accelerates that evolution; initially, Cloud databases serviced consumer applications. These early applications put a priority on read access because the ratio of reads to writes was very high [1][2]. Delivering high-performance read access was the primary purchase criteria; however, this is changing.

User generated content, particularly in the form of social networking, have placed somewhat more emphasis on updates. Reads still outnumber writes in terms of the ratio, but the gap is narrowing. With support for transactional business applications, this gap between database updates and reads is further shrinking. Business applications also demand that the Cloud database be ACID compliant: providing Atomicity, Consistency, Isolation and Durability.

C. Paper Structure

The rest of the paper focuses on what are the factors to be considered to have a database running in the Cloud. In Section II, we study the shared-disk and shared-nothing architecture and decide which one is the best to support our database in Cloud. Section III acquaints you with the Cloud and defines the term DBaaS (Database-as-a-Service). As Cloud uses multiple machines it is required to have them in sync, Section IV provides you with a brief overview of few synchronization algorithms. We list various Cloud storage providers in Section V and also compare few of them. We describe the various consistency properties that guarantee transaction in a database and also mention why one is popular over the other when it is used in Cloud. Section VI describes

Cloud services provided by Amazon and we also recommend based on your requirement which one is good for you. Section VII looks at one possible security implementation for your database in the Cloud. We then conclude our paper with the future scope of database in the Cloud in Section VIII and conclusion in Section IX.

II. Shared-Disk vs. Shared-nothing

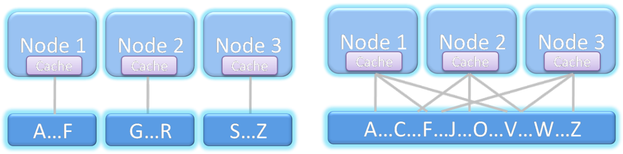

Figure 1: Shared-nothing and shared-disk architecture [3]

Features	Shared-Disk	Shared-Nothing
Load Balancing	Not bound to particular CPU	Not supported
High-Availability	Downtime reduced because no partitioning; Reroute of request to different servers	Repartitioning requires downtime
Data Consistency	Always consistent	Choice between performance and consistency
Parallel Processing	Effortless	Problematic
Scalability	Limited by messaging overhead between nodes; mitigated by elasticity	Impractical due to large data transfers between nodes
Elasticity	Add/remove nodes without impact to underlying data	Inefficient
Server Utilization	Requests redistributed—no CPU is idle	Requests bound to a specific server—not every server utilized
Operating Costs	Pay less for maintenance; pay more for Cloud-specific software	Human intervention with system increases costs

Table 1: Feature comparison between shared-disk and shared-nothing architecture [2]

Many databases used today are running on a shared-nothing architecture; however, most Cloud databases require the ability to dynamically scale the entire system. If we maintain the partitioning system used in a shared-nothing environment, there may not be any increase in performance gains from adding more servers. Using middleware is an option for a shared-nothing environment but performance gains are still cancelled by data shipping across nodes. More information can be found in [2].

Large corporations such as Amazon, Google, and Facebook have implemented their own persistence engine that puts more focus on replicated tables in order to support their scalability needs. However, these are not a true replacement for a real database in a corporate Cloud Computing environment.

Partitioning of data has been eliminated by the shared-disk architecture—ideal for Cloud databases. With shared-disk architectures, a group of servers are connected to a Network Attached Storage (NAS) or Storage Area Network (SAN) [4]. Thus, the data is available to all servers, which eliminates the need for partitioning. Consequently, a shared-disk architecture increases our performance by adding servers.

A. Multitenancy

Multitenancy in software architecture is the practice of having a single instance of software serving multiple clients—or tenants.

Currently many existing setups use a shared-nothing architecture where each client has their own instance. This is being phased out in favor of multitenancy because network speeds and shared storage are no longer limitations for performance.

In a Cloud environment, multitenancy is helpful in reducing costs by providing multiple customers with the same CPU and accessing the same disk and same data simultaneously. Various issues what people are looking today is not just security in multitenancy but how to obtain trust-worthiness.

B. Benefits of Shared-Disk

Few benefits observed in [1] are listed below:

- **Fewer number of servers** – Separate nodes specifically for fault tolerance are not needed.
- **Lower hardware costs –** Able to use multiple low-end servers rather than a single high-end server.
- **No downtime for simple maintenance** – Simply disconnect a node to perform hardware upgrades.
- **High-availability** – No downtime to promote a slave to master due to master-master configuration.
- **Data always consistent** – A single copy of the data is used; proper locking mechanism is required.

III. Cloud

A. The Cloud Defined

Before we continue, it would be pertinent to define what we mean by the phrase, "Cloud Computing"; this relates to the foundation of the entire paper. We will be modeling section III based on the study in [5], which discusses various aspects of Cloud Computing.

Cloud Computing is characterized by off-site access to shared resources in an on-demand fashion; it refers to both the services delivered over the Internet and the hardware and software in the data centers that provide the services.

In short, Cloud Computing encapsulates both Software as a Service (SaaS) and X as a Service (XaaS)—where X may be replaced with Platform or Infrastructure. The reason for interchanging Platform with Infrastructure as a Service is due to the different levels of control given to customers of Cloud Computing; further details are discussed later in the Public Clouds subsection. The commonalities between SaaS and XaaS may be easier to comprehend if you were to associate the terms with Utility Computing.

The primary client of Cloud Providers, we believe, is to be SaaS developers who will have dynamic hardware requirements—as is the nature of their service.

B. Cloud Providers

Services are provided at each layer: SaaS, PaaS, and IaaS. Database.com [6] (from salesforce) and SmallThought's dabbledb [8]—recently acquired by Twitter—are the popular database providers running as SaaS. For PaaS, the largest providers are Google AppEngine [9][10], Microsoft Azure [11], and 28msec's Sausalito [12]. The three biggest IaaS providers today are Amazon [13], GoGrid [14] and Mosso [15]. The type of services offered by some providers is shown in the figure below:

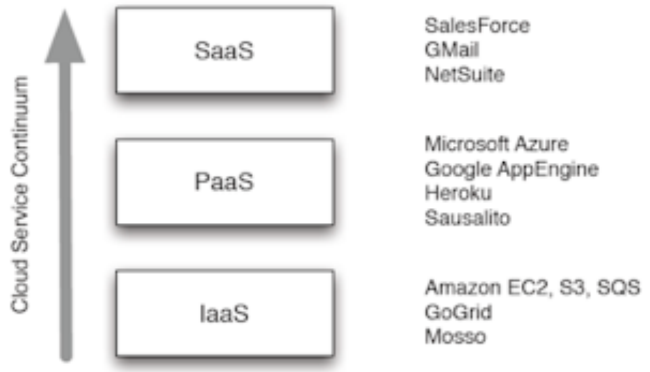

Figure 2: Cloud Service Continuum [16]

C. Benefits

The clear benefits of moving to a Cloud infrastructure are agreed to be as follows due to the fact that a Cloud uses shared-disk configuration [5]:

- **On-demand** – Acquiring resources when needed.
- **Pay Per Use** – Billing in a utility manner.
- **Scalability** – Supporting the illusion of infinite computing resources.
- **Maintenance and Fault Tolerance** – Responsibility shifted to the service provider from our IT department.
- **Reduce Costs** – Capital can be shifted away from acquiring the infrastructure.

D. Database as a Service

Database as a Service (DBaaS) may be thought to be a special case of SaaS where a client may rent out a database instance for personal use. As such, you are not required to specialize in database management or even the database query language (incase of using NoSQL). There are various vendors offering DBaaS listed in [7].

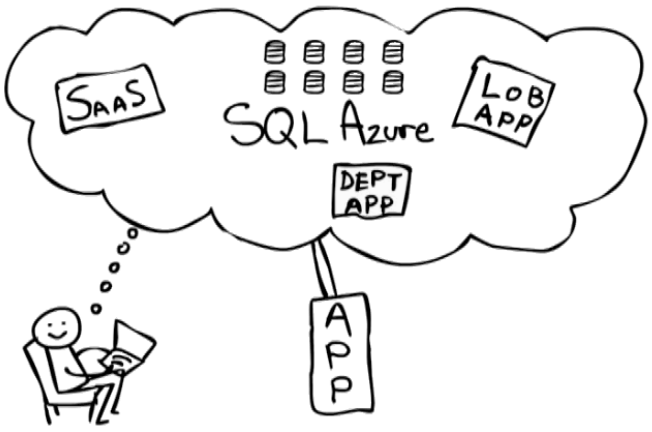

Figure 3: Visualized DBaaS [17]

A visual representation of how a Cloud Solution (SQL Azure here) may be used as a DBaaS is shown above.

E. Private Clouds

Generally, referring to the term Cloud Computing does not include the notion of Private Clouds. However, this paradigm shift could benefit large companies who are willing and able to host their own datacenters throughout the country.

In fact, the same issues exist whether in a Private or Public Cloud: hardware not being efficiently utilized, network bandwidth sporadically idling and peaking, etc. With regard to these problems, a Private Cloud should be able to provide resolution. For many employees, their workstations are hardly used efficiently—often for word processing or even remotely accessing another server within the intranet. In these instances, CPU cycles are wasted on providing employees with powerful machines. Rather, moving those cores and disks to the Private Cloud while giving employees a dumber device may better utilize company resources.

Still, there are also obstacles that cannot be solved by hosting one's on Private Cloud such as adding new nodes. For a client to the Public Cloud, it is mentioned that there is a layer of abstraction where everything is believed to be infinite. That is, a client may be able to simply request and reserve as many CPUs or petabytes of storage capacity, relatively hassle free. It is pushed to the Cloud Provider to verify that the physical resources are available for them. Because, in a Private Cloud, the enterprise is both the client *and* provider, there is no simple web form to add eight additional petabytes of disk storage to a particular Fibre Channel over Ethernet (FCoE) datastore. However, such is the cost of maintaining one's network and is nothing new to deal with whether or not a Private Cloud is implemented.

Of course, there are use cases where this may not be feasible, but there is never one solution to best fit everyone [5]. This should, as everything else, be a business decision with both pros and cons weighed.

F. Public Clouds

With Public Clouds, there is a clear separation between the client and provider of Cloud services that is difficult to mistake. In this scenario, we have a Cloud Provider who, presumably, already has a datacenter setup for their business uses and applications. The client, on the other hand, may vary from an individual up to any sized enterprise that finds the Cloud to be an economical alternative. Consequently, by entering in a Service Level Agreement (SLA) with the Cloud Provider, the client may be able to offset some responsibilities and costs away from himself.

Utility Computing gives us a new business model where customers pay for only whatever resources they use from the Cloud, rather than paying for their own hardware and associated support. Consequently, a client of the Cloud may start small and over time grow their required resources if need be due to an illusion of infinite resources being available to them. An additional benefit is that the client need not worry about the IT hassles such as installation, maintenance, upgrades, etc. These are especially useful for startups and even medium sized companies where it is economically unfeasible to build a private datacenter. Rather, they are now given the option to acquire storage or CPU cycles from the Cloud for a cheaper rate—allowing them to invest their capital elsewhere, such as development or marketing. Such elasticity is a great benefit for those who are able to wisely take advantage of such a feature.

Due to the distinct spheres of both clients and providers of Cloud Computing, the differences between Platform as a Service, PaaS, and Infrastructure as a Service, IaaS, are more appropriately discussed under the umbrella of Public Clouds.

With PaaS, the client is simply given a platform for software development—to create the SaaS applications. This layer of abstraction allows for a clear distinction between the logical layer and more physical layer with regard to hardware. IaaS, on the other hand, provides users with a lower level view by giving them privileges upward from the kernel. Simply, PaaS could be thought of as Google AppEngine while Amazon EC2 more accurately exemplifies IaaS. Interestingly, Microsoft Azure sits in the middle ground between the features of AppEngine and EC2.

When a user looks for a Cloud Provider, it is vital to consider what requirements and scalability issues may need to be addressed in the future. With a PaaS, the additional layer of abstraction between clients and the virtualized hardware allows for a more streamlined scaling of hardware resources. IaaS, on the other hand, would be a bit more difficult to scale because the client has much lower level access to the virtualized resources. As such, it may even be argued that it is up to the client to worry about synchronizing and parallelizing the work between multiple hosts while PaaS gives that responsibility to the Cloud Provider.

G. *Clouds to Stay*

Cloud Computing is not an entirely new concept [5]. However, this iteration of dumb clients has the best potential to become a permanent paradigm shift due to the hardware and networking technologies available to end-users today. It is now realistic to have smaller, portable devices that remotely connect to central servers from nearly any street corner.

IV. Distributed computing Algorithms

With distributed computing, in general, there are similar problems that must be addressed if we are to have a stable and usable system in a production environment. In this section, we will go through an overview of only a few of the algorithms available to help with consensus and voting, which were found in [16][18]. Note that there are other topics such as time synchronization that are also problematic, but beyond the scope of this paper.

A. *Byzantine Agreements Protocol with Oral Messages*

The Byzantine Generals Problem is a fitting analogy for distributed computing systems relying on asynchronous communications. If *all* loyal generals attack, victory is at hand. If *none* of the loyal generals attack, no losses will be incurred. If *some* of the loyal generals attack, the war will be lost. Therefore the only acceptable solutions we have is for all loyal generals to act in a similar fashion.

To do this, we will use the simplest of Byzantine Agreement Protocols in order to demonstrate the general idea: the Oral Message Algorithm.

Assume there are N total generals, with M traitors. For this to work, N >= (3M + 1) must be true. In the first round, all nodes broadcast their value to (N – 1) nodes. In the second round, each node sends all the values they received to (N – 2) nodes. This continues until each general has the values of every other general. Finally, take the majority value as the action to perform.

B. *Quorums for commits*

As with many algorithms used in distributed transactions, the goal here is for a consistent view in the system. With quorums, each node may belong to a different voting district. This is important because a process must retrieve an entire quorum before being able to commit to the system.

To acquire a quorum when voting districts exist, we must have two full districts that will have a non-empty intersection. Let us assume we have the following grid quorum, below:

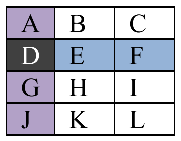

Figure 4: Example of Quorum

We notice that the nodes {A, D, G, J} are all part of the same voting district just as {D, E, F} are associated in yet another district. Because D overlaps, acquiring these two districts is a valid quorum to guarantee consistency.

A simpler train of thought is to imagine a quorum as a simple majority. If every transaction required a simple majority—in this case, 7 of 12 nodes—then there should always be at least one up-to-date node retrieved from the quorum.

C. *Quorums for replicas*

Another use for quorums, aside from voting in commit protocols, is to vote in replica control protocols. In this scenario, we have three variables:

N	Replication factor
R	Read quorum
W	Write quorum

Generally, each replica in the group of N is maintained on a separate machine. As such, the read and write quorums must decide upon the amount of replicas that must be acquired before proceeding with the respective operation. More specifically, if we wish to read or write to the replicas we must first temporarily acquire a lock on them—the exact amount being R or W depending upon the operation.

Now, as long as R+W > N holds true, we can guarantee that the latest modifications will be read back if inquired—thus, consistency. If we were to lower the values to R+W <= N the same guarantee cannot be made.

V. STORAGE FOR CLOUD DATABASES

A. *Overview*

Here we will provide a simple example of the Cloud Services architecture described in [16]:

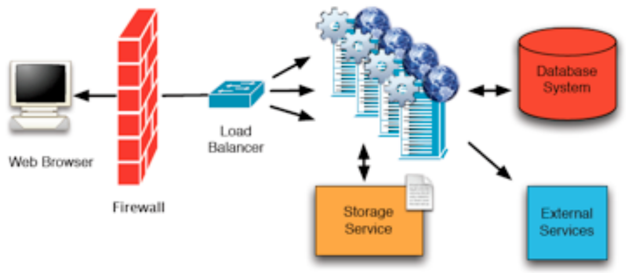

Figure 5: Cloud Web Architecture

In the image above, different Cloud Service Providers could potentially provide the Database System, External Services, and Storage Services—which sit behind the computing machines as shown above. When a request comes in, the Load Balancer will redirect it to a computing machine that may then use the other Cloud Services.

B. *CAP Theorem*

The CAP theorem states that there can only be two of the following three attributes that can be held true at any one time [16]:

- Consistency
- Availability
- Network Partitioning

Because we cannot afford to lose support of network partitioning, we must choose between consistency and availability. As such, we must design the Cloud Service to be placed somewhere between consistency and availability.

C. *ACID vs BASE*

Due to the CAP theorem described above, we must choose between consistency and availability. However, if we focus on consistency while holding ACID to be true, we will lose the availability property of our database. Hence, another model has been proposed: BASE. BASE stands for Basically Available, Soft state, Eventual consistent.

A major difference between ACID and BASE is that ACID remains pessimistic which forces consistency at the end of every operation. BASE, on the other hand, is optimistic which allows it to accept inconsistencies [16]. The Eventual consistent aspect of BASE guarantees only those clients will eventually have a consistent view of the database.

D. *Storage Solutions*

There are many existing storage solutions—some of which are open source. It should be noted that there are also open source solutions available such as Cassandra [19] and CouchDB [20]. A few commercial storage providers are listed below along with their internal storage solutions.

Service Provider	**Internal Implementation**
Amazon	Dynamo [24]
Google	BigTable [9], Megastore [10]
Microsoft	Microsoft SQL Server
Yahoo	Apache Hadoop [21]

Table 2: Commercial Storage Service provider

E. *Amazon S3 vs. Amazon EBS*

Because each commercial service uses their own proprietary setup and algorithms, we will be examining Amazon's storage services S3 [22] and EBS [23] in this section.

Features	**Simple Storage Service (S3)**	**Elastic Block Storage (EBS)**
Accessibility Limitations	Any availability zone	Single EC2 instance, same availability zone
Consistency Model	Eventual consistency	Session consistency
Latency	Higher	Lower
Replication	Multiple datacenter	Single datacenter
Storage Model	Key-value	Volumes
Search Capabilities	Key-range scans	OS dependent

Table 3: Comparing S3 and EBS

F. *Amazon Dynamo*

Internally, Amazon uses Dynamo, which is not a relational database [24] because it is difficult to create redundancy and parallelism using traditional database system. As the database grows, it becomes the bottleneck of the system; adding more web servers does not help in mitigating the scalability problem. Hence, Amazon needed a better solution: one that is scalable, guarantees high availability over consistency, and maintains SLAs.

Dynamo stores information for retrieval using a simple query model through which all objects are stored and identified via a unique key without using a relational schema; the data is not broken into tables. Additionally, Dynamo uses eventual consistency due to the CAP theorem [24] as discussed in Section V.B.

Dynamo provides a simple *put* and *get* interface that relies on a key, context and object relationship. The context is metadata about the object and is used to validate updates. Identical physical nodes are organized into a ring; virtual nodes are created and mapped onto physical nodes as well so that hardware can be swapped for maintenance and failure.

The partitioning algorithm, specifying which nodes will store a given object, is by far the most complicated piece of the system because it automatically scales as nodes enter and leave the system. Every object is replicated to a number of nodes and are updated asynchronously which may then result in multiple copies of the object in the system with slightly different states. The discrepancies are reconciled after a period of time ensuring eventual consistency. The diagram below shows the high level view of service-oriented architecture at Amazon.

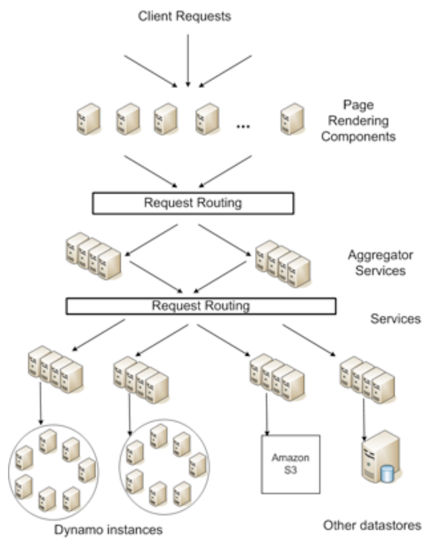

Figure 6: Service Oriented Architecture of Amazon's platform [24]

G. *Compromises of Databases in the Cloud*

A few differences between the traditional RDBMS hosted on a local system and a Cloud Database are provided below:

- No ACID property
- Transactions not supported
- CAP theorem limitations

VI. Amazon Web Services

The following sections are services that are available to the consumer through Amazon.

A. *Computing Service*

Elastic Computing Cloud (EC2) by Amazon allows a client to rent computing resources such as CPU and RAM by the hour —cost varies depending on the configuration desired [13]. When rented, a virtual machine is provided to the customer. The implementation of EC2 is actually running on top of the XEN hypervisor, which is open source.

In the case of failure, EC2 does not support VM migrations. Unless the state of the VM was stored elsewhere—on S3, for instance—the information is lost. Another benefit for running EC2 in conjunction with S3 is that the latency between the two is miniscule. However, there is no restriction of choosing a third-party storage service.

Having no VM migration denies the fact that your instance is always available, 100% of the time. Thus, if hardware upgrades were to be done your VM would be down for the duration.

B. *Database*

If we were to store indexes on the storage service, the costs are not worth the performance increase; hence, many Cloud Providers provide an indexing service as a feature such as SimpleDB [25][26] or Microsoft SQL Azure [11]. Azure, however, currently has a 50 GB size limit per database [11]. It is worth noting that if the Cloud Provider does not have a suitable indexing service, then self-managed indexing is always an option; however, this would not be suitable for a database on the Cloud.

- SimpleDB

SimpleDB by Amazon supports point and range queries on structured data [26]. The data model of SimpleDB relies on Domains, Items, Attributes, and Values. Domains may be described by any collection of items that share a similar attribute-value pair. These are analogous to SQL tables. Items are similar to SQL rows, Attributes related to column headers, and finally Values may be thought of as the individual cell values in the Attribute columns.

With SimpleDB, Amazon allows for two different consistency models to be used: Eventually Consistent Reads and Consistent Reads [26]. With Eventually Consistent Reads, we have the focus of performance in mind while Consistent Reads cares much more about consistency. Thus, you would generally choose to move from the default of Eventually Consistent to Consistent Reads when your application requires reading of up-to-date material.

SimpleDB does not provide full transactions, but provides us with *conditional puts* and *deletes* [26]. A conditional *put/delete* will only *put/delete* if the given condition is satisfied. Additionally, complex queries such as join, group by, etc. are not supported.

Amazon has suggestions for who would be most interested in SimpleDB. We paraphrase these below:

- Those who care only about indexing and searching
- People who want to offload administration
- Scalability in mind
- High availability in mind

There is also a limitation in SimpleDB where attributes are not allowed to be greater than 1024 bytes [16]. As such, it is recommended to store the data in S3 and simply point to them from SimpleDB if the attribute size is large.

- RDS

Relational Database Service by Amazon provides clients with the option to migrate their already existing RDBMS setups to the Cloud without much hassle. Doing so will give us advantages such as replication and backups done by Amazon.

Once integrated with RDS, the replication can be setup to reside in a different Availability Zone where the replica is in a separate physical location with independent infrastructure [27]. An Availability Zone is a groups of servers based on geographical locations.

Amazon provides the following scenarios for suggested clients of RDS:

- If a relational database is required for a new or existing application.
- Offloading the infrastructure while maintaining a MySQL relational database.
- Those interested in scaling their database with a simple API call.

- AMI

Amazon Machine Images are preconfigured virtual machines provided by Amazon to be deployed on EC2 instances [28]. The purpose of this is to allow the clients to more easily have a particular operating system and application setup.

AMIs includes support for the following databases [28]: IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, Aster Data and Vertica.

The potential customers who would be interested in using pre-configured AMIs are the following:

- Those who desire a wide variety of database engines.
- Clients who need complete administrative control over the database server.

C. *Storage*

More information on storage may be found above in Section V.E and Section V.F regarding Amazon S3, EBS and role of Dynamo in the Amazon's storage services.

VII. Security

A. *Relational Databases*

Maintaining a secure database is not simple but it is beneficial to have when the database has critical elements and is placed in the Cloud. The key challenge here is to perform SQL queries over the encrypted data in an efficient manner.

B. *Onion Security*

The idea here is to have multiple layers of encryption similar to onion routing. A paper proposed by [29][30] suggests four different layers of encryption:

- **Randomized Encryption (RND)** – This provides the maximum security.
- **Deterministic Encryption (DET)** – Provides weaker privacy; however, allows the server to process equality conditions.
- **Order-Preserving Encryption** – Even lower encryption strength, but enables inequality checks to be performed along with sorting operations.
- **Homomorphic Encryption (HOM)** – Provides the least security of the four, but allows more operations [addition, subtraction etc.] over encrypted data.

Graphically, this could be shown as:

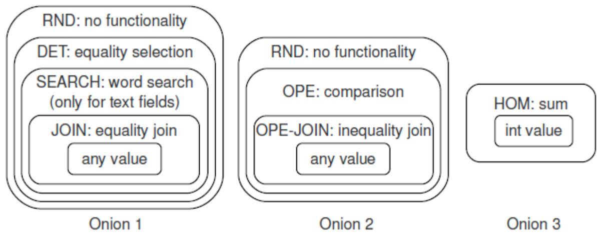

Figure 7: Onion layers of encryption and classes of allowed computation [29]

As for the end-to-end solution, we provide a sample implementation below:

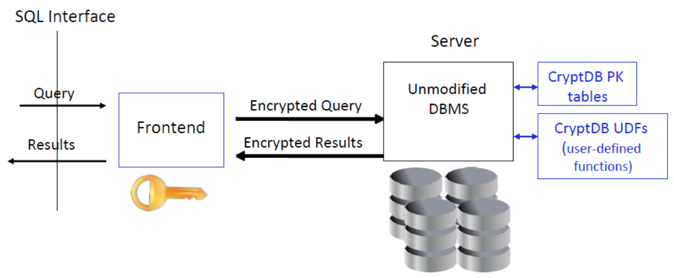

Figure 8: Implementation of Onion security

The above figure describes the flow in the process when a query is fired which is executed over the encrypted database. Depending on the key, various restrictions are imposed on what operation is to be allowed by the query. We will not discuss Onion Security Implementation as it is out of scope of this paper. More information can be obtained in [29][30].

VIII. Future Works

Currently, the security aspects of hosting databases in the Cloud have not been thoroughly examined. As such, we believe there is more work to be done—such as upholding the integrity of your data. This is also known as trusted multitenancy and was a topic of presentation at the RSA 2011 Conference [31].

Additionally, the ACID properties would be beneficial for any transaction with a database in the Cloud. At this time, there are only solutions such as Amazon RDS and Microsoft SQL Azure which provides ACID property but with few limitations either with the size of the database or performance.

While many shared-disk is generally thought to be the go-to solution for operating a Cloud, we believe that shared-nothing may something to offer because there still remains a debate between the two.

From the views of the Cloud Provider, migrating to low power consumption hardware such as Intel Atom chips would help to lower their operating costs even more.

Furthermore, rather than being limited to a single Cloud Provider, we may want to query multiple Clouds using a virtualization layer which performs all the optimization and data acquisition for the end-user. This way, the developer does not need to know the specifics to every Cloud database; rather, he must only learn the query language of the virtualization layer.

IX. Conclusion

Cloud Computing, one of the fastest growing areas in the computer industry, is being adopted by many IT departments to provide on-demand services for their clients' requirements—thereby minimizing maintenance, hardware, and other miscellaneous costs. The success of Cloud Computing is due to these cost savings, rather than the technology behind its implementation. The advantages of hosting web-based applications are compelling but there are some drawbacks in the sense that different providers have different interfaces and functionality, which makes the applications harder to port from one database to another.

This paper presented an overview of what database in the Cloud is, and describes how databases are hosted at different levels. We have also described why it is beneficial to use shared-disk vs. shared-nothing architecture for database in cloud. In shared-disk architecture, there are multiple nodes with only one copy of the disk that all the nodes access; hence, we need distributed computing algorithms to synchronize communication and timing between the nodes.

We then moved onto Cloud Storage, where data is stored on multiple virtual storage systems that are hosted by third-party vendors rather than having dedicated storage systems. The data stored in Cloud Storage solutions is accessible through various application programming interfaces (API) or through a web-based user interface. Accessing these systems through an API or query language, such as XQuery, was beyond the scope of this paper. You may find more information in [18]. Our focus was on AWS because they offer comprehensive Cloud-oriented services; they were also pioneers in successfully marketing the cloud.

Security, due to its importance in today's world, was examined in order to alleviate the worries of clients who have critical information that they rely on for their living. Privacy and confidentiality, however, is not only a database issue—this is the big worry with most enterprises migrating to the Cloud.

References

[1] Mike Hogan, *Cloud Computing & Databases - How databases can meet the demands of cloud computing,* Whitepaper, ScaleDB Inc., November 14, 2008, available from www.scaledb.com/pdfs/CloudComputingDaaS.pdf

[2] Mike Hogan, *Database Virtualization and the Cloud - How database virtualization, cloud computing and other advances will reshape the database landscape*, White paper, ScaleDB Inc., December 10, 2009, available from www.scaledb.com/pdfs/Cloud_Databases_WhitePaper2.pdf

[3] Ben, *Shared Nothing v/s. Shared Disk Architectures: An Independent View,* Nov 24, 2009, available at http://www.benstopford.com/2009/11/24/understanding-the-shared-nothing-architecture

[4] Carlo Curino, Evan Jones, Yang Zhang, Sam Madden, "Schism: a Workload-Driven Approach to Database Replication and Partitioning", *The 36th International Conference on Very Large Data Bases,* September 13-17, 2010, Singapore. Proceedings of the VLDB Endowment, Vol. 3, No. 1

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia , "*Above the Clouds: A Berkeley View of Cloud Computing*" Technical Report EECS-2009-28, EECS Department, University of California, Berkeley.

[6] Steve Bobrowski, *The Future of Databases is in the Clouds,* Whitepaper, SalesForce.com Inc., available at http://wiki.database.com/page/The_Future_of_Databases_is_in_the_Clouds

[7] Senior Developer Evangelist for Salesforce.com, *Database as a Service (DbaaS) Product Directory,* available at http://dbaas.wordpress.com/database-as-a-service-dbaas-product-directory

[8] SmallThought Inc now Twitter Inc, dabbledb, San Jose, CA, available from http://dabbledb.com/demo

[9] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data", *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Seattle, WA, November, 2006.

[10] Jean-Michel Leon, Yawei Li, Alexander Lloyd, Vadim Yushprakh, "Megastore: Providing Scalable, Highly Available Storage for Interactive Services" Google, Inc., *5th Biennial Conference on Innovative Data Systems Research (CIDR '11),* January 9-12, 2011, Asilomar, California, USA

[11] David Chappell, "Introducing the Windows Azure Platform", Whitepaper, Microsoft Corporation Inc., Redmond, WA, OCTOBER 2010, available at http://www.microsoft.com/windowsazure/whitepapers

[12] 28msec Inc., Sausalito, available at http://www.28msec.com

[13] Amazon.com Inc., Elastic Cloud Computing (Amazon EC2), San Jose, CA, available from http://aws.amazon.com/ec2

[14] GoGrid Inc., GoGrid, San Francisco, CA, available at http://www.gogrid.com

[15] Rack Space Inc., Mosso, San Antonio, Texas, available at http://www.rackspace.com

[16] Microsoft. *What is SQL Azure Database*. Streaming Video. 2011. https://www.microsoft.com/en-us/sqlazure/database.aspx.

[17] Holliday, JoAnne. *Distributed Computing Lecture Notes*, Santa Clara University, 2009.

[18] Tim Kraska, *Building Database Applications in the Cloud,* Technical Report Diss. ETH No. 18832, Swiss Federal Institute of Technology, Zurich, 2010, Available at e-collection.ethbib.ethz.ch/eserv/eth:924/eth-924-02.pdf

[19] Lakshman, Avinash and Malik, Prashant, "Cassandra: a decentralized structured storage system"*SIGOPS Oper. Syst.* Rev. 44, 2, Pg. 35-40, April 2010, 0163-5980

[20] An Apache Project, CouchDB, available from http://couchdb.apache.org/

[21] Dhruba Borthakur, The Hadoop Distributed File System – Architecture and Design, Whitepaper, Apache, November 9, 2006, available from hadoop.apache.org/common/docs/r0.18.2/hdfs_design.pdf

[22] Amazon.com Inc., Simple Storage Service (Amazon S3), Seattle, WA, available from http://aws.amazon.com/s3

[23] Amazon.com Inc., Elastic Block Store (EBS), Seattle, WA, available from http://aws.amazon.com/rds

[24] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels. "Dynamo: Amazon's Highly Available Key-value Store" Whitepaper, Amazon, Inc. http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf

[25] Amazon.com Inc., SimpleDB, Seattle, WA, available from http://aws.amazon.com/simpledb

[26] Lew, *Amazon Web Services – SimpleDB Overview,* April 22nd, 2009, available at http://clouddb.info/2009/04/22/amazon-web-services-simpledb-overview.

[27] Amazon.com Inc., Relational Database Service (Amazon RDS), Seattle, WA, available from http://aws.amazon.com/rds

[28] Amazon.com Inc., Running Databases on AWS, Seattle, WA, available from http://aws.amazon.com/running_databases

[29] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan, "CryptDB: A Practical Encrypted Relational DBMS", *Waiting to be published, internally published in Department of Electrical Engineering and Computer Science at MIT.*

[30] Carlo Curino, Evan Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Samuel Madden, Hari Balakrishnan, Nickolai Zeldovich, "Relational Cloud: A Database Service for the Cloud", *5th Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2011.

[31] Saif Khan, Martin Pueblas, "Trusted Multi-Tenancy", *RSA Conference 2011*, SPO2-302, Feb 2011 available at https://cm.rsaconference.com/US11/catalog/catalog/catalog.jsp