ISSN:0975-9646

Cloud Offerings: A Systematic Review

Dr. Reema Ajmera
Asst. Professor
School of Computer Sciences and Applications,
JECRC University, Jaipur, India

Rudra Gautam
Application Architect
Vertex Business Services
Gurgaon, India

***Abstract*— Open source is one of the core foundations of cloud computing. Early pioneer of the cloud utilized the freely available, freely distributable model of open source to power their vision and deployments- achieving a level of scale at a bare-bones cost that had never been seen in the history of computing. In this paper first we discuss cloud basic than we move towards the offerings provided by various service models.**

***Index Terms*—Elasticity, pay-per-use, CAPEX, OPEX, ASP**

I. INTRODUCTION

This paper describes, how cloud services are offered using cloud computing as well as the different types of computing clouds and their specifics. With the advent of cloud computing the scenario of using and accessing of resources i.e. servers, application and storage is changed significantly. Gartner defines Cloud as *a style of computing in which scalable and elastic IT-enabled capabilities are delivered as a service to external customers using Internet technologies.* Nowadays these are available on demand or pay-per use basis. This feature separates Cloud from traditional application service providers or pure virtualization environment.

Elasticity – Elasticity is the degree to which a system is able to adapt to workload changes by provisioning and de--provisioning resources in an autonomic manner, such that at each point in time the available re-sources match the current demand as closely as possible.[1]

Applications that experience a higher load during a certain time of the year, for example, can request more resources only during these periods. Especially, for development and test purposes, this property of clouds can make their use very profitable.

Pay-per-use – no monthly charge for resource use is applied. Costs only arise for resources during their usage times. Therefore, no long-term upfront investments (CAPEX) in IT resources are required anymore. Instead, only the operational costs (OPEX) of these resources arise.

Standardization – Cloud provides image-based system management standardizes the used hardware software stacks that are used in cloud applications through the use of hardware virtualization.

II. CLOUD SERVICE MODELS

Cloud Service Models describe the different ways to offering resources served as a service by the many cloud service provider. On the basis of the portion of the application stack that is controlled by the provider, one differentiates between Infrastructure, Platform, Software, or Composition as a Service (IaaS, PaaS, SaaS, CaaS respectively and many more).

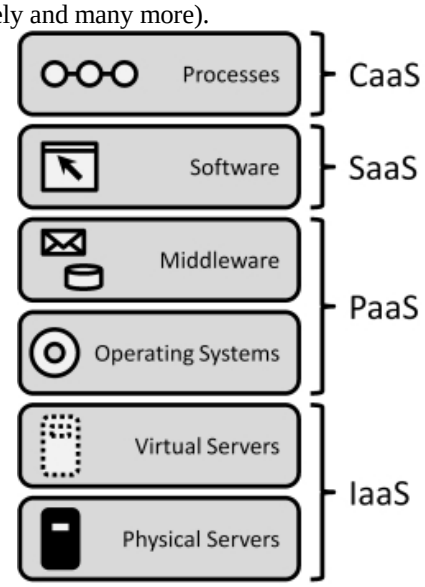

Fig.1- Cloud Application Stack

Cloud Computing is offered in various models which includes:

Infrastructure as a Service (IaaS)

IaaS (Infrastructure as a Service) is when the responsibility of the equipment is outsourced to the Service Provider. The Service Provider not only owns the equipment but will also be responsible for its running and maintenance, where the consumer will be charged on a 'pay as you use' basis and not concerned about infrastructure. IaaS is often offered as a horizontally integrated service that includes not only the server and storage but also the connectivity domains. In IaaS, Access control is added to an elastic infrastructure and the resource management is extended to isolate users from each other. The monitoring component collects additional information to support pay-per-use billing.

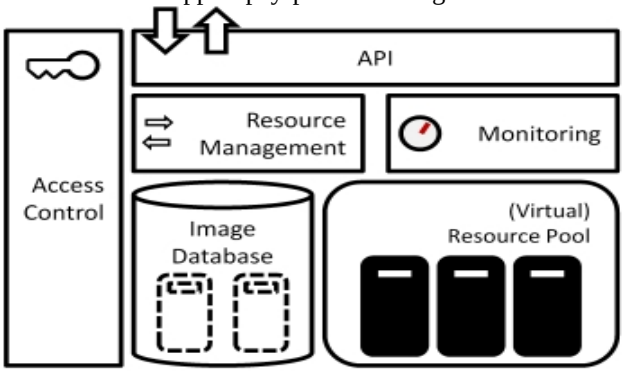

Fig.2- IaaS Working Model

It controls authentication of user and control their usage of the API during the management of (virtual) server images and the starting/stopping of (virtual) servers. Furthermore, the monitoring component is extended to support billing based on accesses to the API as well as the amount of used resources.

Infrastructure as a Service may be part of the offerings that form a *public*, *private*, or *hybrid cloud*. The first and still the most dominant provider of IaaS is Amazon EC2.

Platform as a Service (PaaS)

PaaS provides the capability for consumers to have applications deployed without the burden and cost of buying and managing the hardware and software, these are either consumer created or acquired web applications or services that are entirely accessible from the Internet. PaaS facilitates immediate business requirements such as application design, development and testing at a fraction of the normal cost.

The platform itself therefore has to be made multi tenant-aware, so that software components cannot access data or functionality and do not influence the performance of other users' software components. Elasticity of the hosted components managed and enabled automatically [3]. To achieve proper functionality an API allows users to deploy software components to a *Platform as a Service* offering, register and configure other platform services for communication e.g., message queues, storage e.g., block storage, and routing e.g., realized in an enterprise service bus.

Accesses to deployed software components and registered platform services are controlled to ensure isolation of users and consumers. The middleware components, such as applications servers, enterprise service busses, and messaging systems are extended to assure equivalent performance to all users. Software components are often created using specific development environments or libraries to ensure certain component properties, such as statelessness for example, to enable a platform controlled elasticity of deployed applications. Billing services, also offered by the platform, are often based on the amount of storage used, number of messages sent, or accesses to the hosted services.

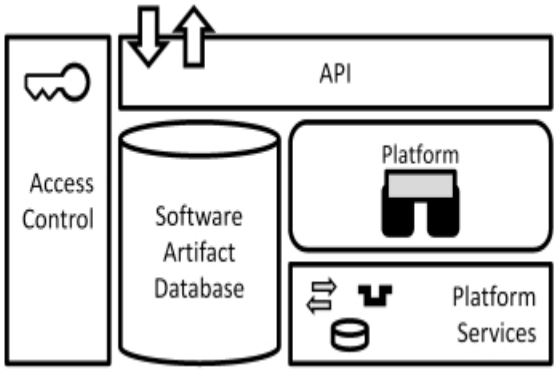

Fig.3- PaaS Working Model

The Google App Engine and Salesforce's Force.com platform are pure PaaS offering where the user is unaware of the underlying infrastructure or its management.

Software as a Service (SaaS)

Software as a service (SaaS) is the ability for a consumer to use on demand software that is provided by the service provider via a thin client device e.g. a web browser over the Internet. With SaaS the consumer has not only no management or control of the infrastructure such as the storage, servers, network, or operating systems, but also no control over the application's capabilities. Culled from what were originally referred to as (ASPs) Application Service Providers, SaaS is a quick and efficient delivery model for key business applications such as customer relationship management (CRM), enterprise resource planning (ERP), HR and payroll and many more.

A user interface or an API is used to access the *Software as a Service*. Access is controlled to ensure the isolation of multiple users while the desired customization is stored in a central database that controls how shared components behave.

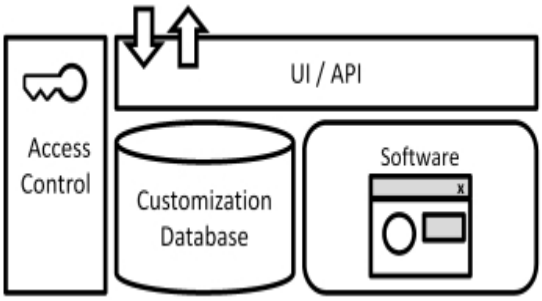

Fig.4- SaaS Working Model

The software provides functionality that is integrated with a user's application that he runs on his own premise. Access to the hosted software is controlled to avoid that users can access other users' data or influence the performance that others experience. The customizations specified by users are stored in a database and are accessed from software components to determine their behavior.

The components out of which the offered software is comprised can be implemented accordingly to *multi-tenancy patterns* to ensure the required multi-tenant awareness.

Software as a Service may be part of the offerings that form a *public*, *private*, or *hybrid cloud.* The first significant provider of SaaS was Saleforce's web-based CRM software [2].

Composite as a Service (CaaS)

Different provider supplied services shall be offered to users that are isolated from each other on a pay-per-use basis. These users shall be enabled to create individual compositions of the provider supplied services to meet their functional and service level requirements.

Individualization of IT services regarding offered functionality and service levels often requires changes to the service implementation and the structure of the hosting environment. A good alignment of user requirements and the offered services is however mandatory to increase the addressable market. The service provider therefore has to integrate functionally equivalent services assuring different service levels and run them in different hosting

environments. However, creating all possible combinations of services to be selected from users is often unfeasible.
Users may create custom compositions of provider supplied services residing on the software, platform, or infrastructure layer. These customizations are again hosted by the provider.

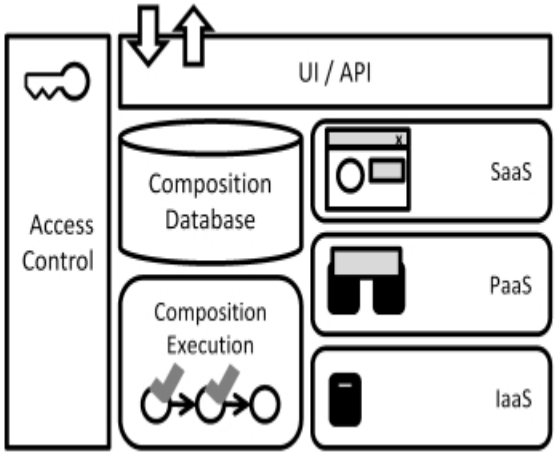

Fig.5- CaaS Working Model

The customer composes services offered by the provider to reflect the functional and service levels that he requires. This composition is uploaded to the provider and stored in a composition database. Access control ensures that the data and compositions of multiple users remain isolated. A user accesses a composition either through a user interface or an API depending on the type of service that is offered by the composition.

A runtime is included to execute the customer specified compositions. Different variations of th`is runtime could be available. For example, a CaaS provider could provide a BPEL or a Java Engine.
CaaS is still an ongoing researchsubject and CaaS is already offered as online platforms that allow modeling and execution of business processes, such as RunMyProcess , Cordys Process Factory , or Intalio's online modeling , Microsoft's process modeling capabilities, named Sharepoint Designer, also form a CaaS offering.
Except above mentioned service models some of other emerging models are as given below:

Monitoring as a Service (MaaS)

Monitoring as a Service (MaaS) is at present still an emerging piece of the Cloud jigsaw but an integral one for the future. In the same way that businesses realised that their infrastructure and key applications required monitoring tools that would ensure the proactive elimination of any downtime risks, Monitoring as a Service provides the option to offload a large majority of those costs by having it run as a service as opposed to a fully invested in house tool. So for example by log`ging onto a thin client or central web based dashboard which is hosted by the service provider, the consumer can monitor the status of their key applications regardless of location.

Communication as a Service (CaaS)

Communication as a Service (CaaS), enables the consumer to utilize Enterprise level VoIP, VPNs, PBX and Unified Communications without the costly investment of purchasing, hosting and managing the infrastructure. With the service provider responsible for the management and running of these services also, the other advantage the consumer has is that they needn't require their own trained personnel, bringing significant OPEX as well as CAPEX costs.

STaaS - Storage as a Service

STaaS is fairly self-explanatory, the basics behind the STaaS service is that the cloud provider rents out storage space on their infrastructure in return for a monthly fee. The reason STaaS can be appealing is due to economies of scale within the service's infrastructure, this leads to providers being able to rent out space at a lower cost than your average business would be able to provide its own storage solutions. Storage as a Service is often seen as an option to solve the problem of offsite storage backup.

SECaaS - Security as a Service

Security as a Service acts as an outsourcing model for computer and network security services. Security services on offer include things such as anti-virus, anti-spyware/adware and intrusion detection. When considering the cost of ownership, SECaaS can be a cheaper alternative to having your own individual or corporate security protection.

DaaS - Data as a Service

Data as a service allows consumers to use a provider's data on demand anywhere over a network connection (usually the internet). DaaS providers both cleanse and enrich the data and then offer it to different users, programs or applications.

TEaaS - Test Environment as a Serivce

TEaaS or Test Environment as a Serivce (or "On-demand test environement) is a model used to enable users to test software. This is done by allowing users to store the software and its associated data in the cloud, this is then accessed through a web browser over an internet connection.

APIaaS - Application Programming Interfaces as a Service

APIaaS - Application Programming Interfaces as a Service - Application Programming Interfaces as a Service is another fairly self-explanatory service, it provides consumers with a service to build and store API's and efficient model of offering.

BaaS - Backend as a Service

Backend as a service is sometimes referred to as MBaaS (Mobile Backend as a Service). This model of cloud computing allows for consumers to link their applications to back end cloud storage while providing other such features including push notifications, internation with social media and user management. BaaS is a comparatively new development within the cloud computing universe, with mainstream BaaS services being offered from around 2011. As market availability is still ongoing process

AaaS - Analytics as a Sevice

Analytics as a Service is a cloud platform that allows users to access powerful analytics tools hosted within the cloud. This is a developing area within the cloud computing world and industry experts are expecting more and more companies within more and more industries to take advantage of this technology.

Anything as a Service (XaaS)

Interestingly XaaS or 'anything as a service' is the delivery of IT as a Service through hybrid Cloud computing and is a reference to either one or a combination of Software as a Service (SaaS), Infrastructure as a Service (IaaS), Platform as a Service (PaaS), communications as a service (CaaS) or monitoring as a service (Maas). XaaS is quickly emerging as a term that is being readily recognized as services that were previously separated on either private or public Clouds are becoming transparent and integrated. Yet an ongoing offering and still available in multi-varients.

III. CLOUD OFFERINGS

So as the term 'The Cloud' finally breaks into the minds of the masses and takes meaning, in this section we describe numerous services that are offered by the Cloud, mature them and enable consumers to fully understand their benefits.

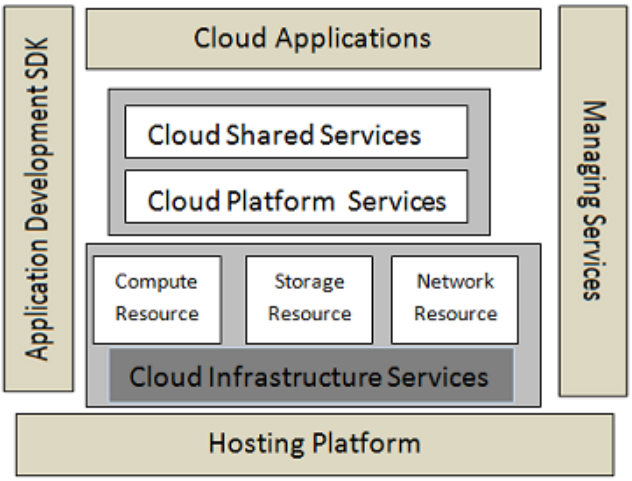

Fig.6- Cloud Offerings Model

a. Cloud Compute Services

Compute services are used by cloud customers to execute their workload. To do so, customers may deploy their own application components on the cloud provider's infrastructure. Different Cloud Service Models may be used in this scope: Infrastructure as a Service – the customer controls the complete operating systems and middleware that the application requires; Platform as a Service – the customer only maintains the application itself. The complete required runtime environment is completely maintained by the cloud service provider.

Elastic infrastructure

The infrastructure must support dynamic provisioning and deprovisioning of resources. This requires the possibility to start and stop preconfigured (virtual) servers and their automatic integration in communication networks. Further, the infrastructure needs to allow the dynamic allocation of memory, processors, and storage etc. as well as the monitoring of these system resources to determine the utilization of the (virtual) servers. This functionality must be offered through an API to be used by atomized management tools and the applications that are hosted by the environment. An *elastic infrastructure* allows the management of (virtual) servers and other resources, such as storage) through and API that offers start, stop, and allocation operations as well as monitoring of resources.

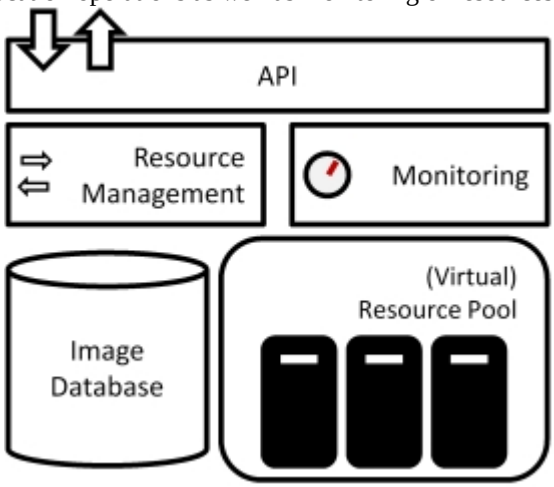

Fig.7- Elastic Infrastructure

An elastic infrastructure supports the dynamic allocation of (virtual) resources that constitute a common resource pool. In case of server resources it allows dynamic starting and stopping of preconfigured (virtual) servers[5]. This is enabled by storing so called server images in an image database. These (virtual) server images contain a description of the hardware configuration, the operation system, and possible additional middleware and software components.

Additionally, an elastic infrastructure contains a resource management component that handles the allocation of physical resources when requests are initiated via the API. Through a monitoring component information about (virtual) resources, such as utilization, may be extracted from the outside.

The elastic infrastructure forms the fundamental basis for cloud computing. It is therefore mandatory for the cloud service models, such as *Infrastructure as a Service*, *Platform as a Service*, or *Software as a Service*.

VMware ESX, Xen, HyperV etc. are classical virtualization environments that offer the functionality of an elastic infrastructure.

Low Availability Computing Node

A computing environment shall be provided for services whose availability is not critical. A large number of commodity (virtual) servers is used that provide the required performance and that provide the extraction of monitoring information.

Fig.8- *Low Availability Computing Node Figure*

The health status of (virtual) commodity servers is monitored so that a server failure can be detected.

Low available compute nodes usually constitute a public cloud. Often, applications have higher availability requirements than what is guaranteed by low available compute nodes. The watchdog pattern shows how an evaluation of the monitoring information and corrective

actions can lead to a higher availability of the system composed of (virtual) commodity servers.
Many virtual servers of public clouds are offered at a low availability. Sometimes, availability is additionally expressed in an uncommon manner. For example, Amazon guarantees an availability of EC2 instances of 99.95% during a service year of 365 days

High Availability Computing Node
A computing environment shall be provided for services whose availability is critical. A high availability compute node is used that is specifically build to provide the required level of availability and performance through internal management functions[5].

Fig.9- ***High Availability Computing Node***

The high available compute node continuously monitors itself and detects failures and performance problems. This information is used to react to faults and performance bottlenecks automatically. Often, this is enabled by redundant internal components that can replace failing ones. Traditional mainframes such as IBM zSeries are built to be fault tolerant and self-healing. They provide many virtualization technologies similar to those used to realize cloud computing.

b. Cloud Storage Services
Storage provides a large number of ways to store data while working on the application. They needed to be flexible to support a wide range of application requirements. Cloud storage services offer centralized cloud-based storage for applications or application components. Especially, if compute services do not offer the required availability, centralized storage is required to integrate several replicas of application components. These methods are employed in *componentized applications* and the *watchdog pattern* specifically addresses availability concerns.

Strict Consistency
A storage offering usually consists of multiple replicas to ensure fault tolerance. It is of major importance that the consistency of the data contained in these replicas is pertained at all times while the performance is of secondary importance.
The highest level of consistency is granted if all replicas are updated if the data contained by them is altered. However, this would mean that the availability of the overall storage solution is decreased drastically. It has to be ensured that it is available even if not all replicas are available, but still the correct version of the data is read. Access only subsets of replicas during read and write operations to increase the availability. The size of these sets guarantee that at least on replica is read with the most frequent version at all times. The used read and write operations are subsumed in an ACID transaction.

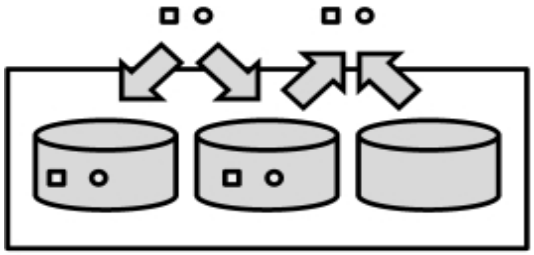

Fig.10- ***Strict Consistency***

Subsets of the available replicas are accessed during read and write operations. Thus, the system is available even if not all replicas are accessible. Strict consistency is guaranteed through the size of the subsets of replicas that are read or written. Considering the overall number of replicas, the number of replicas access during read , and those accessed during write , it is ensured that holds true for every read and write operation. Therefore, each read accesses at least one more replica than the previous write, ensuring that at least one replica is accessed with the most current version. The values for and are usually fixed at design time and reflect the different requirements on read and write performance. If write performance shall be increased, then the number of replicas accessed during a write is decreased and those during read increased, for example.
Some storage solutions allow the specification of consistency behavior on a per request basis. Therefore, critical information can be retrieved following strict consistency; less critical information is retrieved granting eventual consistency.

Eventual Consistency
A storage offering usually consists of multiple replicas to ensure fault tolerance. It is of major importance that the availability of the data contained in these replicas is increased while the consistency of the data is of secondary importance.
Assuring consistency among many (geographically distributed) replicas reduces the availability of the storage offering since it makes it dependant on the network over which replicas are updated. To handle possible partitions in this network, traditional replica models read and write a certain number of replicas to guarantee that at least one replica is read with the current version. Depending on the priority of read and write accesses the ratio of replicas that have to read or written can be adjusted. Therefore, either during read, write, or both multiple replicas must be accessible which reduces the availability of the overall storage offering. In certain scenarios consistency of data can however be relaxed to reduce the number of replicas to be accessed by operations. This increases the availability of the storage offerings since it is more robust regarding network partitioning.
It has however to be assured that changes to replicas are eventually propagated to all replicas. If some replicas are unavailable during a write the changes need to be stored at a different persistent location and need to be executed once the replica is accessible again. Additional challenges arise when replicas in different network partitions are changed

independently and have to be merged once the network is not partitioned anymore.

Eventually consistent data storage allows reducing data consistency to increase availability and performance, since the impact of network partitioning is reduced and fewer replicas have to be accessed during read and write operation.

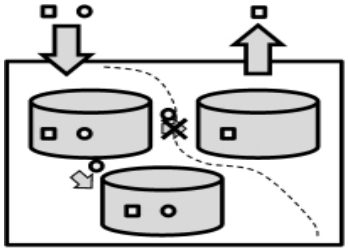

Fig.10- Eventual *Consistency*

Eventually consistent databases increase the availability during network partitioning at the expense that inconsistent data can be read under certain conditions. This is achieved as follows:

While strictly consistent databases ensure that always at least one of the current version is read, eventually consistent databases allow that obsolete versions may also be read. This increases the availability of the storage offering since only one replica has to be available to successfully execute a read operation. Using this database model it can also be avoided that writing multiple replicas has to be executed as a distributed two-phase commit (2-PC) to guarantee ACID behavior. Instead replicas are updated asynchronously via transactional massages, for example. This guarantees that after a network partitioning problem is eventually resolved, changes to data are propagated to all replicas. A "consistency window" specifies how long this update via messages takes in absence of network partitioning. However, using eventually consistent storage also demands a certain data scheme and / or operation that are idempotent, so that changes to partitioned replicas can eventually be merged. Amazon SimpleDB uses two consistency models, strict consistency and eventual consistency.

Relational Data Store

An application uses a central database for storing data elements and performs complex queries on them.

Applications often access a database remotely and perform complex operations on the contained data elements. Queries are sent to the database to retrieve matching data elements. The more precise these queries can describe the required elements, the less stress is put on the network transporting data elements and on the application processing them.

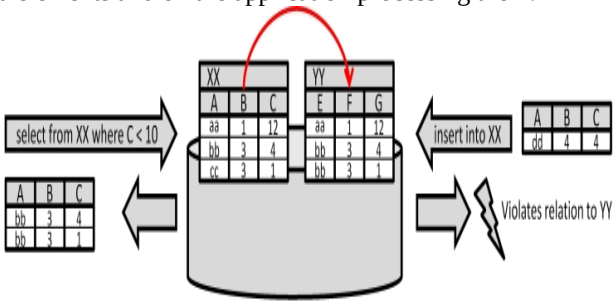

Fig.11- Relational Data Store

The relationships between attributes of elements in one table and elements in another table are expressed in a database schema that describes the structure of the database tables. Whenever a data element is created, altered, or deleted it is verified that the relations described in this schema are fulfilled.

Complex queries can be send to the database that express ranges of and conditions on data element attributes. Only the attributes that match the specified conditions are returned to the querying application. SQL is a common query language for this purpose.

A relational data store is offered by traditional data base systems, such as IBM DB2, Oracle RDBMS, MySQL , or Microsoft SQL Server. These can be also realized on top of an IaaS cloud.

Blob Storage

A distributed application needs to manage large data elements, such as virtual server images or videos, which are too large for traditional databases.

In a distributed application data elements must be made available to all application components and to distributed users. Access to the data needs to be performed in a standardized fashion and access control has to be established.

Organize the data elements in a folder hierarchy similar to a traditional file system. Give each data element a unique identifier that can be used to access it over a network. Also, establish access control mechanisms.

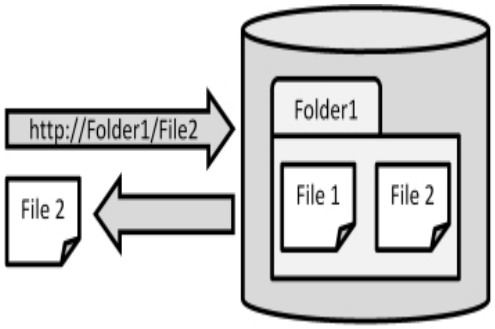

Fig.12- Blob Storage

The data elements are stored centrally and in hierarchical folders. Within each folder every data element is given a unique name. The combination of the position of an element within the folder structure, its unique name, and the service address form the address at which the element can be accessed. This access is enabled using established technologies, such as FTP, Rest over HTTP, SOAP over HTTP etc.

The number of hierarchy levels in the directory structure is sometimes reduced (see S3 below). Also, sometimes automatic distribution about multiple geologically distributed replicas is offered to guarantee locality of data.

Traditional Web an FTP servers function in a very similar fashion. Amazon's S3 services offers similar functionality but only allows folders without subfolders (called buckets).

In Windows Azure similar functionality is provided by Windows Azure Storage , a service that subsumes a *message oriented middleware*, *NoSQL storage*, and *block storage*.

Block Storage

Servers forming a distributed system shall access a central high available storage as if it was a local drive.

Resources in clouds are often unreliable (*low available compute nodes*). Therefore, the data that they access locally shall in fact be stored in a high available central data store. This way, if a server fails the data is not lost, but a new server can be started to use the secured data.

Offer data elements in a central storage that can be accessed by distributed servers and integrate them as local drives.

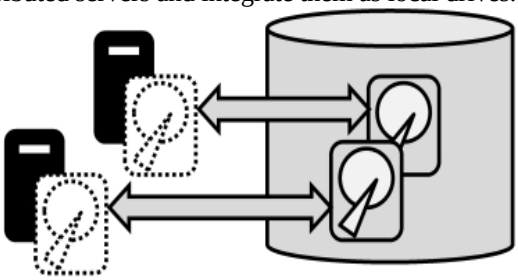

Fig.14- Block Storage

Server can integrate centralized files into their system and treat them as if they were regular hard drives. Thus, they are formatted with a certain file system and accessed through the operating system of the server. Often, these servers are virtualized, since the integration of files as hard drives is a basic functionality of virtualization software.

Block storage is offered as a component of the Windows Azure Storage Service and Amazon EBS [4].

NoSQL Storage

Data storage shall be provided that is distributed among many resources and that frequently has to handle data structure alteration.

Traditional relational database management systems are often hard to scale-out, for example due to dependencies between tables arising from foreign keys. The complexity of join operations, if data from remote systems has to be combined, forms an additional challenge. Those database systems are usually configured to utilize the resources of a central server or cluster optimally. However, cloud based applications usually do not have access to centralized, high performance servers but instead to a large number of distributed, commodity systems. These applications need to handle very large amounts of data and also need to be adjusted to new user demands flexibly. Therefore, a database solution is required that focuses on scaling out rather than on optimizing the use of a single resource and that can adjust flexibly to changes of the data structure.

Use a schema-free storage solution, with limited query capabilities to enable extreme scale-out through easy data replication.

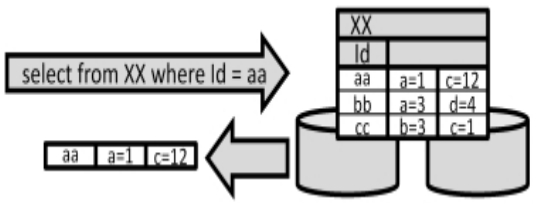

Fig.15- NoSQL Storage

The resulting NoSQL databases either do not support any schema at all or a very limited one. Often, they only allow querying a single index parameter. This allows them to be distributed among very many resources and the structure of the handled data remains extremely flexible.

This of course adds additional complexity to the application that uses such a data base. Changes to the implicit structure of the data need to be handled by the application as well as consistency checking. Also, no join operations are supported, which leads to generally more data returned to the application initiating the query. Traditional databases tried to reduce this amount of data through sophisticated join operations and conditional expressions. When using a NoSQL database this has to be implemented on the application level.

To increase availability and performance even further, NoSQL storage solution often display *eventual consistency*. But *strict-consistency* solutions are also possible. In the case of Amazon's SimpleDB this can even be specified on a per-request basis.

c. Cloud Communication Services

Applications running in the cloud rely on different communication services, because of the distributed nature of cloud resources. These communication methods are used cloud internal, for example, to exchange messages between application components. Further, communication services enable the integration of clouds with each other or with traditional data centers into *hybrid clouds*.

Message-Oriented Middleware

Different applications usually use different languages, data formats, and technology platforms. When one application (component) needs to exchange information with another one, the format of the target application has to be respected. Sending messages directly to the target application results in a tight coupling of sender and receiver since format changes directly affect both implementations. Also, direct sending tightly couples the applications regarding the addresses by which they can be reached.

Connect applications through an intermediary, the message oriented middleware, that hides the complexity of addressing and availability of communication partners as well as supports transformation of different message formats.

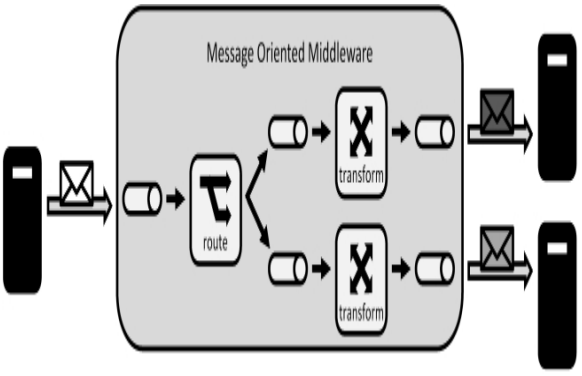

Fig.16- Message Oriented Middleware

Communication partners can now communicate via messages without the need to know the message format used by the communication partner or the address by which

it can be reached. The message oriented middleware provides message channels (also referred to as queues). Messages can be written to these queues or read from them. Additionally, the message oriented middleware contains components that route messages between channels to intended receivers as well as handle message format transformation.

A message oriented middleware is often used, if data needs to be exchanged in a responsive manner. If larger amounts of data need to be exchanged applications can be integrated through file transfer or by sharing a common database.

Exactly-once delivery

Communication partners exchange messages via a message oriented middleware.

Even though the use of *reliable messaging* avoids duplication of messages in general, implementation specifics may still lead to message duplicates. This affected by the design decision how long to wait for a system to recover eventually from its persistent storage after it became unavailable. Sometimes the timeliness of messages demands resending them. Also, in some scenarios, especially regarding business to business interaction, r*eliable messaging* may not be available at all. Under these conditions duplicate messages need to be handled.

Associate messages with unique identifiers and use filters to delete duplicates.

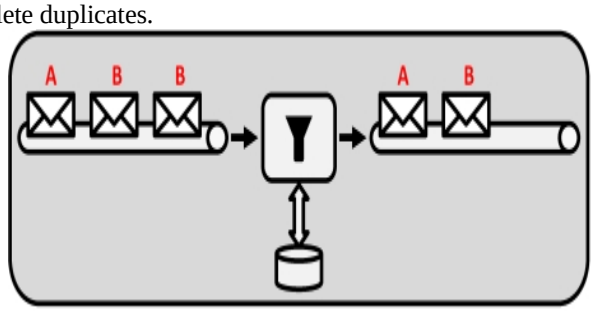

Fig.17- Exactly-once delivery

Whenever a message is created it is associated with a unique identifier. This is used by a filtering component on the message path to delete duplicates. It does so by storing the identifiers of messages it has already seen. The identifiers of messages passing through this filtering component are then compared to the identifiers that have been recorded to identify and delete duplicates. A central design decision is the size of the list that stores message identifiers, because it dramatically affects the robustness of the solution and its performance. Often, messages are associated with a time frame in which they are valid to limit the size of message identifier lists.

The filtering of messages can also be part of the receiver instead of being implemented in the message oriented middleware. This would then form an *idempotent components*.

The mentioned message filter is described as a separate pattern in.

At-least-once delivery

Under some conditions receiving duplicate messages is uncritical. For example, if a database, like an organization employee directory, is queried using messages the re-execution of a query does not affect the state of the database. Therefore, the additional overhead to avoid the occurrence of message duplication during their transmission shall be reduced while still guaranteeing that a message is received.

Acknowledge that messages are received and retransmit the messages that have not been acknowledged.

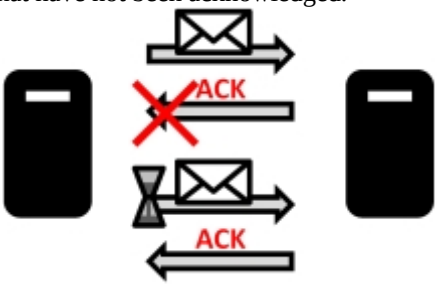

Fig.18- At-least-once delivery

The receiver of messages sends special acknowledge messages to the sender. If the sender does not receive such an acknowledgement message in a given time frame it retransmits the message. Thus, messages, which are lost due to communication errors, are still received eventually. However, duplicate messages can occur, for example, if an acknowledgement message is lost.

To reduce the communication overhead, acknowledgement messages can be sent either after each individual message or after an agreed upon number of messages.

If sender and receiver of messages communicate via a queue the acknowledgment is not sent to the sender of the message but to the queue. A receiver removes the messages from the queue and acknowledges when he has finished processing it. Instead of deleting the message, after its removal from the queue, the messaging system keeps it in persistent storage. If the acknowledgement is not received after a certain time period, the message is put back on the queue.

Today, all cloud messaging services guarantee the described at-least-once behavior. Amazon SQS and Windows Azure Storage service both implement the version of this pattern where messages are put back on queues automatically if receivers fail to acknowledge their processing.

Reliable Messaging

When messages are exchanged in distributed systems, errors can occur during the transmission of messages over communication links or during the processing of messages in system components. Under these conditions it shall be guaranteed that no messages are lost and that messages can be eventually recovered after system failure.

Message exchange during communication partners is performed in under transactional context guaranteeing ACID behavior.

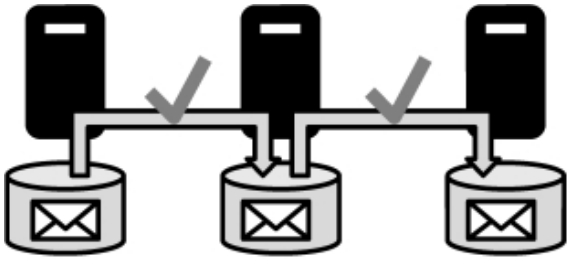

Fig.19- Reliable Messaging

The message transfer from one communication partner to the other is performed under transactional context. Especially, this transaction subsumes the operation performed to store the messages in persistent storage. Thus, if an error occurs during message receiving, sending, or processing the transaction can be compensated transferring the overall system back to a correct and consistent state.
Sometimes not every communication partner has access to persistent storage. In this case the receiving of a message is contained in one transaction with its processing and the sending of another message to a communication partner that has access to persistent storage.

In the cloud, there are several messaging systems that can be accessed as a service, such as Amazon SQS or the queue service part of Windows Azure Storage. The cloud-based offerings however differ regarding the granted delivery model. They offer at-least-once delivery.

REFERENCES

[1] Nikolas Roman Herbst, Samuel Kounev, Ralf Reussner, Elasticity in Cloud Computing: What It Is, and What It Is Not,USENIX Association ,10th International Conference on Autonomic Computing (ICAC '13),pp-23-27
[2] Salesforce.com Inc.: CRM & Cloud Computing. Available at: http://www.salesforce.com/
[3] http://aws.amazon.com/what-is-cloud-computing/
[4] http://aws.amazon.com/rds/
[5] http://www.cloudcomputingpatterns.org
[6] Amazon.com: Simple Queue Service. Available at: http://aws.amazon.com/sqs/
[7] Microsoft: Windows Azure Storage. Available at: http://www.microsoft.com/windowsazure/storage/

AUTHOR

Reema Ajmera received M.Sc.(CS), M.Tech.(CS) and Doctorate (Comp. Sc. Eng.) Degrees in 2004, 2007, and 2013 followed by worked as software developer for three years in a MNC and then joined Jaipur National University(JNU) as Assistant Professor, School of Computer and Systems Sciences in 2007. Research interest includes Agent Oriented Software Engineering, Cloud Computing, Agile Testing and Software Re-Engineering. Active blog provider and technical reviews on many research topics.

Rudra Gautam received Diploma(CS), BCA, MS(CS), MSC(IT) in 2004, 2006, 2008, followed by working in multiple MNCs and holding 12+ years of experience in software designing, architecture and development. Expensive experience in software product development and architecture since 2001. Started professional career after diploma and continuing higher education parallel to work. Research interest in Cloud Computing, Cloud Management and Network Architecture. Technical evangelist and blog writer. Expertise in enterprise portal, digital commerce and CDN implementation.

.