Cloud Computing and Big Data

Josh Felmeden

October 4, 2021

Contents

1 Overview

1.1 Economic Driving Factors

Cloud computing works by charging someone to use a service for a certain amount of time. If, for example, you let someone use your computer, you might charge them for their usage (if you were a real meanie). How that would be worked out is the *operation expenditure* (Opex) and the *capital expenditure* (Capex, cost of the computer). Combining these, we calculate the *total cost of operation* and calculate the cost per day of the lifespan of the computer (in this example). So, if you used the computer for an hour, you would owe the person a 24th of this daily cost.

Now, this might come down to half a penny, but of course, this doesn't exist any more, so we might charge a whole penny instead. This seems like a marginal profit, but in terms of percentages, this is a 100% increase. The fundamentals of this are how cloud computing generates so much income.

The attraction of this is that the users of the services do not have to pay the Capex, and simply pay opex for the rental of the service.

1.2 Normal Failure

Failures in these systems are to be expected. Say the servers you are buying are guaranteed to have a 99.999% 3-year survival rate ('five-nines reliability'). This is good because there is a very high chance that this remains. Now, if you buy 10 of these servers, the probability that you have all of the servers still working is only 99.99%. Taking this to the extremes, if you buy 500,000 servers (this is standard practise for a lot of the big servers these days), the probability that all of them are still working is a measly 1%. Essentially, failure is something that should be normalised.

Modular data centres are used a lot in big data centres. A unit may be left in the shipping container, and this container is removed or added as a container, meaning that if one module fails, another can just be replaced, meaning that the whole centre doesn't collapse.

In the current climate, the modular centres are considered a whole working unit, and this way of thinking was mobilised by a group of Google engineers.

1.3 Blank as a service

- **SaaS: Software as a service**
  - End user application software that is remotely delivered over the internet.
  - Adobe is an example of this (Adobe Creative Cloud)
  - Used to be bought off the shelf as a CD
- **PaaS: Platform as a Service**
  - Developer application software (middleware) functionality is remotely accessible
  - Might provide a particular combination of OS, web-server, data-base and scripting
  - Popular instance is the free 'LAMP stack' (Linux, Apache, MYSQL and PHP)
  - Used to be dominated by *Google*

- **IaaS: Infrastructure as a Service**
  - IT infrastructure almost always virtualised and remotely accessible.
  - Virtualisation software allows one physical server to be used by multiple users, each on a virtual machine. If one crashes, the others keep running.
  - Used to be dominated by *AWS* (although this is now a much bigger thing).

While Google and Amazon dominated their respective fields, both of these companies have expanded into the other fields. This is very complicated, but the key thing is that both Google and Amazon offer both PaaS and IaaS.

In around 2015, Amazon created **FaaS: function as a service (aka 'serverless')**:

- No server processes visibly running. Pay only for the time spent executing a function
- UNlike PaaS, scale out without increasing number of servers.
- Amazon Lambda is the best known example, although both Google and Microsoft have answers to this.

1.4 Impacts of Cloud Services

Cloud services have revolutionised business in many ways. It is now possible to do tasks that would previously require access to high performance machines. Instead, it can be sent to a big data centre, and this usage is just charged as rental.

Interoperability is a big issue. **Vendor lock-in** is a concern for a lot of clients, and this simply means that once a client is locked into a vendor, it becomes financially or practically unviable to switch to another supplier. This led to an attempt to develop a sense of unity between cloud companies, where companies created the *Open Cloud Manifesto*. A lot of big companies signed up to this, but a lot of the top dogs didn't sign up for this (unsurprisingly, Google and Amazon). This manifesto seemingly no longer exists as an original. As a consumer, this is bad and worrying because vendor lock in is very possible.

2 IaaS and AWS

2.1 Amazon Simple Storage Service

Amazon S3 is a cloud-based persistent storage. It operates independently from other Amazon services. The simple refers to the features, not that it's simple to use. You can store data in the cloud. You also don't store files, you store *objects*, and these are kept in buckets. Objects have a size limit (5Tb) and a max size on a single upload is 5Gb. All buckets share the same namespace, so no sub-buckets.

It's very easy to use; just use a web GUI that is similar to AWS. It also has a command line interface and has scripting. Default storage can be selected (geographically).

S3 is accessed via API, either by SOAP (xml) or REST (http). Wrappers are available to abstract the

API for programmers.

This is just the storage, so now we will look at the computing of data in the cloud.

2.2 Amazon Elastic Compute Cloud (EC2)

This is a remotely accessible virtual network of virtual servers. Usually, EC2 is run with S3 providing the storage.

A single EC2 virtual server, with the chosen OS etc, is an instance. An instance is instantiatied from an Amazon Machine Image (AMI)

- One AMI can be cloned n times to create n instances
- You can build your own by cloning an AMI from your local server
- Or, Amazon have a bunch of prebuilt AMIs that you can choose from

EC2 dynamically assigns a unique IP address to each instance, and this IP can be reassigned, perhaps to someone else. The IP can also be static (also known as an Elastic IP address) at a cost.

EC2 instances run in availability zones (AZs), grouped into regions. AZ is similar to a single data centre, guaranteeing an area has 99.95% uptime.

2.2.1 Usage

Some basic API routes as S3, command-line or a bit of GUI too:

- Amazon's own AWS web console
- Various EC2 plug-ins for browsers
- Third-party cloud management tools

Some AMIs are junk or malware, however, so be careful when selecting this.

There are three types of storage:

- Ephemeral local storage in the instance (dies with instance)
- Persistent cloud (S3)
- SAN-style Elastic Block Storage (EBS)
  - Allows user to create volumes from 1Gb to 1Tb
  - Any number of volumes may be mounted from a single instance

S3 is slow, medium-reliable, but super-durable. Never loses data, so is good for DR backups. Instance storage is simple and cheap, but speed can be really poor. EBS is high on everything, but is complex and costly.

There is some autoscaling based on matrics:

- **Cloudwatch**: automated monitoring of EC2 instances. Reveals many statistics such as CPU

utilisation, disk reads/writes and network traffic. Aggregates and stores monitoring data that can be accessed
- **Auto Scaling**: dynamically adds or removes EC2 instances based on CloudWatch metrics. You define conditions upon which you want to scale up or down your EC2 instances. Auto scaling automatically adds or removes the specified amount of Amazon EC2 instances when it detects that the conditions have been met.
- **Elastic Load Balancing**: automatically distributes incoming application traffic across multiple EC2 instances. Better fault tolerance. Elastic Load Balancing detects unhealthy instances within a pool and automatically reroutes traffic to healthy instances until the unhealthy instances have been restored. Customers can enable ELB within a single AZ or across multiple for consistent application performance.

2.3 AWS Simple Queue Service SQS and Architecting for Scale-out

SQS is reliable, loosely-coupled fault-tolerant storage and delivery of messages. It can be between any clients or computers connected to the internet, and senders and recipients do not have to communicate directly. No requirement that either side be always-available or connected to the internet.

A **message** is up to 256Kb of text-data, sent to SQS and stored until it is delivered. A **queue** serves to group related messages together.

SQS is accessible to clients on any HTTP-enabled platform. Messages are stored redundantly over multiple data-centres truly distributed. Unfortunately, this brings about a few down-sides:

- Message retrievals may be incomplete
- Messages may not be delivered quickly (2-10 seconds)
- Messages may be delivered out of order
- Messages may be redelivered

2.3.1 Mitch Garnaat's Monster Muck Mashup

This is a service that converts AVI videos to mp4 using:

- 'Boto' Python interface to AWS
- S3 to store the video files
- EC2 to do the conversion processing
- SQS for inter-process communication

Uses AWS for *scalability* (scale-out not up, not just buying new resources, adding cheap machines to improve ability.)

The basic steps are:

1. Upload a bunch of video files to a S3 bucket
2. For each file, add a msg to SQS input queue
3. On the EC2 instance, repeat this until input queue is empty:

a) Read message MI from input queue
b) Retrieve from S3 the video VI specified in MI
c) Do the conversion creating VO
d) Store VO in S3
e) Write message MO to SQS output queue
f) Delete MI from input queue

This really illustrates how good AWS is at scaling software. It is good because any number of clients can connect to the bucket. If this bucket is 100% full, it doesn't matter, because additional instances can all talk to the same buckets and queues, so the workload is met.

2.4 AWS simpleDB and AWS Relational Databases (RDB)

2.4.1 Amazon SimpleDB

This software provides:

- Reliable storage of structured textual data
- Languages that allows you to store, modify, query, and retrieve data sets
- Automatic indexing of all stored data

SimpleDB provides 3 main resources:

- **Domains**: Highest-level container for related data *items*: queries only search within one domain
- **Items**: a named collection of *attributes* that represent a data object. Each item has a unique name within the domain; items can be created, modified, or deleted; individual attributes within an an item can be manipulated
- **Attributes**: an individual category of information within the item, with a name unique for that item. Item has one or more text string values associated with the name

The downside to this type of storage is that it really does only do one data type (textual). If, for example, you wanted to store pi, you would need to store it as a character string ('3', '.', '1').

There is no Database **schema**, meaning if you mistype something, the database will just accept it as a definition, leading to some unfortunate results. It is not a traditional relational database management system:

- SimpleDB items are stored in a hierarchical structure, not a table
- SimpleDB attribute value max size is 1Kb
- SimpleDB data is all stored as text
- The query language is really basic
- SimpleDB is distributed, so data consistency may suffer due to propagation delays

2.4.2 Amazon Relation Database Service

There is also a relational database system, possibly as a response to Azure. You can set up, operate and scale a full MySQL RDBMS without having to worry about infrastructure provisioning, software maintenance, or common DB management tasks, like backups.

The processing power and storage space can be scaled as needed with a single API call and you can fully initiate fully consistent database snapshots at any time. Can import a dump file to get started. Each DB instance exports a number of metrics to CloudWatch including CPU utilisation, etc.

2.5 Availability Zones

Availability zones are clusters of independent data centres that are up to 20 miles apart. They are interconnected using low latency and enable fault isolation and HA.

Choosing which region to use comes down to a few reasons:

- Data sovereignty and compliance
  - Where are you storing user data?
- Proximity of users to data
- Services and feature availability
- Cost effectiveness
  - Each region has differing costs

High Availability (HA) is the ability to minimise service downtime by using redundant components. It also requires service components in at least two AZs.

Fault tolerance is the same as HA, but also the ability to ensure that no service disruption by using active-active architecture meaning that all components are active all the time. This is of course a lot more costly that just having HA.

IaaS may have HA, but FT unlikely. PaaS will usually have HA, but some services offer FT.