Cloud Computing (UE18CS352)
Unit 1

Aronya Baksy

January 2021

1 Introduction to Cloud Computing

- **Cloud Computing**: A model for enabling **ubiquitous**, convenient, **on-demand** network access to a **shared pool of configurable computing resources** that can be **rapidly provisioned and released** with minimal management effort and minimal interaction with service providers.
- Resources available on the cloud include networks, servers, storage, applications and services.

1.1 Characteristics of Cloud Infrastructure

- **On-demand Service**: Self or auto-provisioned resources in short time (may be compute, storage or platform) with no need to interact with IT personnel and wait for approval.
- **Broad Network Access**: Resources should be accessible from any platform (mobile or desktop device, running any OS). Most cloud service providers use the internet for this purpose.
- **Resource Pooling**: Ability to share a single hardware resource between multiple clients. This allows more users to concurrently access the service. It is commonly achieved using Virtualization.
- **Rapid Elasticity**: Easy and fast increase/decrease of the number of resources deployed, based on the current load or other criteria.
- **Measured Service**: Consumer pays only for the resources used by their application (eg: SalesForce charges proportionally with number of customers using the service).

2 Computing Paradigms

- **Centralized Computing**: All the compute resources (storage, memory, CPU etc.) are held in one central location, tightly coupled and shared among all clients. They are accessed from terminal machines (eg: some datacenters, supercomputers)
- **Parallel Computing**: Multiple processors are either tightly coupled (shared memory) or loosely coupled (distributed memory). Inter-Processor communication is done via shared memory or message passing.
- **Distributed Computing**: Multiple autonomous compute nodes that each have their own private memory and communicate via a network. Message passing is used as the mechanism for this communication.

2.1 Grid Computing

- A grid is a system that **coordinates resources** that are not subject to central control, using **standard**, **general purpose** and **open protocols/interfaces** to deliver a **non-trival** quality of service.
- Grids have the ability to handle heterogeneous infrastructure. Trust and security between resources pooled from different organizations on the grid is maintained using resource sharing agreements.

- The end goal of grid computing is to allow computational power to be offered as an utility (like electricity).
- The following are the benefits of grid computing
    1. Exploit underutilized resources
    2. Load balancing between resources
    3. Virtualization of resources at an enterprise level, enable collaboration across VOs
    4. Creation of data grids for distributed storage or compute grids for distributed computing on multiple nodes.

2.1.1 Virtual Organization

- A Virtual Organization (VO) forms the **basic unit** for enabling access to shared resources.
- The key technical problem addressed by grid technologies is to enable resource sharing among mutually distrustful participants of a VO who may/may not have any prior relationship and enable them to solve a common task.

2.1.2 Layered Architecture of Grid Computing

- **Application Layer**: Application running on the grid
- **Collective**: Implement a variety of sharing behaviours with directory, brokering, community authorization and accounting services, as well as collaborative services.
- **Resource**: APIs for allocation of resources as well as secure, negotiation, monitoring, control, accounting and payment for operations on a single shared resource.
- **Connectivity**: Protocols for inter-node communication and authentication of these communications.
- **Fabric**: Provide physical resources (compute, storage, network resources, catalogs) or logical resources (distributed file system, compute cluster) whose access is mediated by the higher-level grid protocols.

2.1.3 Gridware (Grid Middleware)

- A type of middleware that enables sharing and management of grid components based on user requirement as well as resource properties
- Functionality of gridware:
    1. Run applications on suitable resources (brokering, scheduling tasks)
    2. Provide uniform high level access to resources via semantic interfaces (Service Oriented Architecture, Web Service architecture)
    3. Address inter-domain security policies
    4. Application level status monitoring and control
- Examples of gridware: Globus (U Chicago), Condor (U Wisconsin), Legion (U Virginia), IBP, NetSolve (for high-throughput and data-intensive scientific applications)

2.2 Cluster Computing

- Set of loosely/tightly coupled compute nodes that can be viewed as a single system.
- Most clusters consist of homogeneous nodes (each node has same configuration), within a small area, connected by a fast LAN.
- Each node in a compute cluster is configured to execute the same task, scheduled and controlled by software.

- One of the features of a cluster is the abiltiy to merge the multiple systems into a **Single System Image** (SSI).
- An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource. SSI is implemented as a middleware layer (in hardware/software) that presents CPU cores/IO devices/Disks as a single unit shared across all cluster nodes.
- Middleware support is needed to implement SSI as well as high availability (HA) which consists of fault tolerance and recovery mechanisms.
- Instead of implementing SSI at many different levels, virtualization is used to create virtual clusters from smaller number of actual nodes.

2.3 HTC and HPC

- **High Performance Computing**: Usage of large compute resources for a relatively short period of time. Such jobs are typically run on a single system consisting of multiple processors that run tasks in parallel.
- Performance of HPC systems is measured in FLOPS (Floating point operations per second).
- **High Throughput Computing**: Usage of large compute resources for a long period of time. It is defined as a computing paradigm that focuses on the efficient execution of a large number of loosely-coupled tasks
- Performance of HTC systems is measured in terms of number of jobs or operations completed per month or year.

2.4 Parallel Computing

- Computation paradigm wherein multiple tightly coupled processors execute smaller sub-tasks within a larger application program.
- These sub-tasks are *independent of one another* and executed at the same time on different hardware units.

2.4.1 Types of Parallelism

- **Bit-level Parallelism**: Focus on increasing processor word size (eg: an 8-bit processor needs 3 cycles to add 2 16-bit numbers but a 16-bit processor takes one cycle).
- **Instruction-level Parallelism**: Parallel execution of multiple instructions from a single program (eg: parallel loops like vector addition can be converted from loop to parallel instruction).
- **Task-level or Thread-level Parallelism**: Independent threads of execution (performing different tasks) that run on separate processing cores.
- **Data Parallelism**: Input data is split into batches, and all batches are processed in parallel. The exact same instructions are applied to each batch of the data.

2.4.2 Technique and Solutions in Parallel Computing

- **Application Checkpointing**: Record current state of all components in the system so that it can be restarted and restored from that point in time
- **Automatic Parallelization**: Automatic conversion of serial code to multi-threaded code that can be used on an SMP machine (Shared Memory multi-Processor)
- **Parallel Programming Languages**: Classified as either using distributed memory (threads use message passing to communicate) or using shared memory (threads use variables in shared memory to communicate)

Parallel Computing	Distributed Computing
Use of single compute node	Multiple compute nodes
Tasks run on multiple cores on a single chip	Tasks run on a network of computers
Shared or distributed memory	Distributed memory only
Processors communicate through a bus	Processors communicate via message passing
Improve system performance	Improve scalability, fault tolerance, resource sharing

3 Cloud Computing Models

3.1 Enabling technologies

- Broadband networks, internet architecture, web technologies (URLs, HTTP, XML/HTML)
- Multi-tenant technology: single instance of software running on a server and serving multiple clients
- Data center technology
- Virtualization technology

3.2 Cloud Service Models

3.2.1 Infrastructure as a Service (IaaS)

- *Definition*: Capability given to the user to provision processing, storage, network and other fundamental computing resources where the consumer is able to deploy and run any arbitrary software platform (can include OSes, application). The consumer has no direct control over the cloud infrastructure but has control over OS, storage, deployed applications and select control over networking components like firewalls.
- Provision of compute and storage resources as a service. Physical resources are abstracted into virtual containers and presented to the user.
- These virtual resources are allocated on demand to the user, and configured by the user to run any software applications.
- IaaS has the greatest flexibility but the least application automation from the standpoint of the user. It allows the user to have complete control over the software stack that they run.
- Building blocks of IaaS are:
  - Physical data centers (large collections of server racks with multiple physical machines inside each rack), managed by IaaS providers.
  - Compute: the ability to provision VM instances with CPU/GPU configs depending on workload. Also provided are auto-scaling and load balancing services
  - Networking: software abstraction of network devices like switches/routers, available typically through APIs.
  - Storage: either block, file or object storage. Block and file storage are the same as found on traditional data centers, but struggle with scaling, performance and the distributed nature of the cloud. Object storage on the other hand is infinitely scalable, accessible via HTTP, works well with distributed systems like the cloud, uses commodity hardware and allows linear growth of performance wrt cluster size.
- Advantages of IaaS:
  - Flexible
  - Control
  - Pay-as-you-go
  - Faster deployment
  - High availability

- Disadvantages of IaaS:
  - Security threats sourced from host or other VMs.
  - Multi-tenant Security, new VM users must not be able to access data left behind by previous users.
  - Internal resources and training, ie. the need to train IT managers in the use of IaaS management.
- IaaS providers: Google Compute Engine, AWS Elastic Compute Cloud (EC2), MS Azure VMs, DigitalOcean Droplets

3.2.2 Platform as a Service (PaaS)

- *Definition*: Capability given to the user to deploy consumer-created or acquired applications created using programming languages or tools supported by the provider. The consumer does not manage OS, storage, networking or compute infrastructure, but controls the deployed applications and maybe the hosting environment.
- Physical hardware and virtualization of the same is controlled by the provider. The provider, in addition to this, also delivers some selected middleware (like a database software).
- The user can configure and build applications on top of this middleware (eg: create a new database and build applications that use this new database).
- PaaS is well suited to users who use commonly available middleware that is also supported by a cloud provider.
- Advantages of PaaS:
  - Faster time to market due to reduced setup and install time for hardware
  - Faster, easier, risk-free adoption of a large variety of resources (in terms of middleware, OS, databases, libraries and components)
  - Easy to develop for multiple platforms (including mobile)
  - Cost effective scalability
  - Allows for geographically distributed teams, and effective product life-cycle management (build, test, deploy, manage, update)
- Disadvantages of PaaS:
  - Operational limitations (lack of control) due to management automation workflows (available on some PaaS providers) that affect provision, management and operation of PaaS systems
  - Vendor lock-in
  - Runtime issues: specific versions of frameworks may not work with the platform, or platform may not be optimized for the frameworks/language used
  - Security: limited control over hosting policies, risks with storing data on cloud servers
  - Integration and customization with legacy services (like data residing on an existing data center) is more complicated and outweighs the cost saving involved in switching to PaaS.
- PaaS providers: AWS Elastic Beanstalk, Azure DevOps, Google App Engine

3.2.3 Software as a Service (SaaS)

- *Definition*: Capability given to the user to use the applications provided by the provider. The user accesses this application through a thin client such as a web browser on their local machine. The user does not manage the underlying cloud infrastructure (OS, network, servers, storage) or even the application capabilities directly, but instead only changes the application specific settings.
- SaaS is a no-programming model (very limited scripting/programming abilities can be provided in order to change app configuration for advanced users).

- Advantages of SaaS:
  - Flexible payment scheme, pay-as-you-go model
  - High vertical scalability
  - Automatic update of software
  - Accessibility over the internet
- Disadvantages of SaaS:
  - Security of data on cloud servers
  - Greater latency in interaction with app, as compared to local deployment
  - Total dependency on internet
  - Vendor lock-in

4 Technological Challenges in Cloud Computing

- **Elasticity** (Scalability): Resource allocation and workload scheduling algorithms are needed to be able to scale up and down effectively.
- **Performance Unpredictability**: Ensuring reliability when resource sharing is involved
- **Compliance**: with privacy rules, ensure security of information stored on cloud. (in India, the SEBI's Clause 49 lays down these rules).
- **Multi-Tenancy**: Sharing of same virtual resource by multiple users can cause concurrency issues which lead to security problems (hence appropriate locking mechanisms are needed).
- **Interoperability**: Application on one platform should be able to incorporate services found on another platform. This is now possible via web services, which however are complex to develop.
- **Portability**: Application should migrate seamlessly from one cloud provider to another without change in design/programming.
- **Availability**: Reliability that is needed for high availability of cloud resources is hard to achieve due to cascading failures (one failure causes other failures in turn, and so on). This high level of availability is achieved using redundancy at the application, middleware or hardware level.

4.1 High Availability in Cloud

- Cloud uses **failure detection** and **application recovery** to ensure high availability.
- **Failure detection**: cloud detects failed instances/components and avoids directing requests to such instances/components. This is achieved using
  1. **Heartbeats**: Each instance/application sends a heartbeat signal periodically to a monitoring service in the cloud. If the monitoring service doesn't receive a specific number of consecutive heartbeats from an instance then that instance can be declared as failed.
  2. **Probing**: The monitoring service sends a probe to the application instance and waits for a response from the instance. If the instance does not respond to a fixed number of consecutive probes then it can be declared as failed.
- Setting a low threshold for number of missed heartbeats/probes can lead to *faster* failure detection but can also lead to *false positives*. Hence there is a trade-off between speed and accuracy of failure detection
- After identifying failed instances, it is necessary to avoid routing new requests to these instances. A common mechanism used for this in HTTP-based protocols is HTTP-redirection.
- **Application Recovery**: Most commonly achieved using **checkpointing**.
- In check-pointing, the application state is periodically saved in some backing store by the cloud infrastructure. In case the application fails, it can be restarted from the most recent checkpoint.
- Checkpointing is also available at the middleware level (eg: Docker).

5 Cloud Deployment Models

5.1 Public Cloud

- The infrastructure is owned by a cloud provider, an entity (individual or company) pays the provider for access to this infrastructure.
- Resources are virtualized into pools and these pools are allocated among multiple clients that are using the cloud provider's infrastructure (multi-tenancy).
- Access to these resources is done using internet and its associated protocols (SSH, FTP etc).
- The factors that make a particular cloud infrastructure public are: resource sharing using virtualization, usage agreements on resources (pay-as-you-go may or may not be present), and management (provider maintains hardware, networking and virtualization at the minimum)

5.1.1 Advantages

- Low cost
- Less need for server management
- Time saving
- Analytics
- Unlimited scalability, greater redundancy and availability of resources

5.1.2 Disadvantages

- Security
- Compliance with security standards and government rules on data security
- Interoperability and vendor lock-in

5.2 Private Cloud

- Utilizing in-house infrastructure to host cloud services for a single organization
- Can utilize hardware at a local site owned by the organization, or can be a **Virtual Private Cloud** (VPC).
- In a VPC the hardware, networking and virtualization infrastructure is hosted by a third party but with additional security and provisioned config for a secure and exclusive network

5.2.1 Advantages

- More control over resources and hardware
- Security and compliance due to additional layers of security
- Customization, the ability to have custom configurations to run proprietary applications

5.2.2 Disadvantages

- Cost
- Under-utilization of resources
- Platform scaling, upward changes in requirements need scaling of physical infrastructure as against simple scaling of virtual instances on a hosted cloud

5.3 Hybrid Cloud

- A mix of data centers maintained by the organization and hosted cloud infrastructure, connected by a VPN
- A hybrid cloud model allows enterprises to deploy workloads in private IT environments or public clouds and move between them as computing needs and costs change
- This gives a business greater flexibility and more data deployment options. A hybrid cloud workload includes the network, hosting and web service features of an application.

6 Distributed System Architecture

- Three main types of models: architectural models, interaction models, fault models

6.1 Architectural Models

6.1.1 Cluster Architecture

- Building of scalable clusters by connecting smaller clsuters using networks (LAN, WAN, SAN for storage devices). The cluster may be connected to the internet via a VPN.
- The OS and its resource sharing/scheduling policies determine the system image of the cluster

6.1.2 Peer-to-peer Architecture

- Every node acts as both client and server, and all nodes are identical in terms of resources and capabilities.
- NO central coordination or database, no global view of the entire system for one machine, and hence no parent-child relationship between nodes.
- P2P systems are self organizing, and peers can join/leave autonomously.
- Distribution of compute and networking workloads among many links and nodes, thus P2P model is the most flexible and general model.

6.1.3 Client-Server Architecture

- Processes running on server nodes offer service to user requests coming from client nodes.
- Client-server model is implemented as a request-response interaction using send/receive primitives or using Remote Procedure Calls (RPCs).

6.1.4 n-Tier Architecture

- Web applications that forward requests to other enterprise services.
- Specific case is the 3-tier model where client intelligence is moved to a middle tier to enable the use of stateless clients. This simplifies app deployment.

6.1.5 Grid Computing

- - A computing grid offers an infrastructure that couples computers, software/middleware, special instruments, and people and sensors together.
- The grid is often constructed across LAN, WAN, or Internet backbone networks at a regional, national, or global scale.
- Enterprises or organizations present grids as integrated computing resources. They can also be viewed as virtual platforms to support virtual organizations.

6.2 Interaction Models

6.2.1 Synchronous Distributed System

- All the components of the distributed system run on a common clock. The features of a synchronous distributed system are:
    1. Upper bound on message delivery time between processes or between nodes
    2. Message delivery is always in order
    3. Ordering of events happens at a global scale due to a shared clock
    4. Lock-step based execution, meaning that similar operations performed by different nodes in parallel complete at the same time, not at different times.
- These systems are predictable in terms of timing behaviour, hence must be used for hard real-time systems.
- It is possible, and safe to use timeouts to detect application or communication errors

6.2.2 Asynchronous Distributed Systems

- There is no shared clock in such a system, each node maintains its own clock. The following are the properties of asynchronous systems
    1. No bound on process execution time, and no assumptions about speed, reliability of individual nodes.
    2. No upper bound on message delivery time
    3. Clock rates between different nodes may change due to the phenomenon of *clock drift*.

6.3 Fault Models

- Definition of behaviours to be undertaken upon the occurrence of a fault.
- Faults can be in both hardware and software, and fault tolerance (ie. predictable behaviour in case of a fault) is essential.
- The following are the types of faults

6.3.1 Omission and Arbitrary failures

- Fail-stop: Process halts, remains halted. Can be detected by outside applications
- Crash: Process halts, remains halted. May not be detected by outside applications
- Omission: Message inserted in outgoing buffer never enters the incoming buffer of the other end
- Send-omission: Process completes send but message never reaches outgoing buffer
- Receive-omission: Process does not receive a message put in its incoming buffer
- Arbitrary (Byzantine): Arbitrary behaviour wrt message send/receive actions, or omissions, or stopping/incorrect actions.

6.3.2 Timing Faults

- Clock drift from real time exceeds threshold acceptable.
- Process exceeds bounds on task completion time or message transmission time

7 Business Drivers for Cloud Computing

- *Cost*: Low upfront cost of hardware, reduced investment in future scalability, reduces costs of resource under-utilization, and reduced management costs
- *Assurance*: Delegation of management responsibility to a cloud provider reduces need for skilled IT admins and departments, while still maintaining high standards of security and availability.
- *Agility*: Faster response to customer requests for new services, due to faster deployment of new services on the cloud. Also changing business requirements can be better handled
- *Flexibility and Scalability*: Easy to expand resources to meet increased workload
- *Efficiency and improved customer experience*: cloud computing allows streamlined enterprise workflows which result in better workplace productivity, and hence faster biz growth

8 REST and Web Services

8.1 Service Oriented Architecture

- A method to develop reusable software components using service interfaces.
- The use of common communication standards is done for easy integration with existing services (standard network protocols like HTTP/JSON, HTTP/SOAP are used to send requests for various operations)
- Each service in an SOA embodies the code and data integrations required to execute a complete, discrete business function. Services are loosely coupled, meaning that no underlying knowledge of the service implementation is needed to use it.
- Two common SOAs are **REST** and **Web Services**

8.2 REST

- REpresentational State Transfer (REST) is an architectural style for distributed systems, used for providing communication standards between APIs over the internet
- REST-compliant systems (aka RESTful systems) are characterized by their stateless behaviour and the separation of concerns between server and client.
- A **safe** REST operation is one that does not modify any data
- An **idempotent** REST operation is one that does not change the state when applied multiple times beyond the first time.
- REST architectural style is based on:

  1. **Resource identification through URIs**: A resource is a target of interaction between a service and its clients. Every resource is identified by a global identifier called an URI. The existence of a globally unique URI provides global interaction as well as service discovery capability.
  2. **Uniform, constrained interface**: The REST interface consists of 4 basic operations
     - GET: Retrieve a resource (S, I)
     - PUT: Add a resource (update if already exist) (not S, I)
     - POST: Create a new resource (make duplicate if already exist) (not S, not I)
     - DELETE: Remove a resource (not S, I)
  3. **Self-descriptive messages**: Messages contain metadata (how to process and other information about the data). In REST paradigm, resources are decoupled from their representations, hence data can be represented in multiple forms as per the client's understanding. Metadata is used for **cache control**, **transmission error detection**, **authentication or authorization**, and **access control**.

4. **Stateless Interaction**: Server and client need not maintain each other's state, and a message can be understood without referring to any past messages (all messages are independent). Statelessness has the benefits of:
   - Client is isolated against changes on the server
   - Promotes redundancy and improves performance due to reduced synchronization overheads

   State is normally maintained (only if needed) through compact and lightweight text objects called cookies.

8.3 Web Services

- A self contained, self describing modular application designed to be accessible by other applications across the internet.
- Web services are designed to support interoperable machine-to-machine communication over the internet. Other applications and web services can discover and invoke a web service and then communicate with it.

8.3.1 Protocol Stack for Web Services

- **Transport Protocol**: transport messages b/w applications over a network (HTTP, SMTP, FTP, Blocks Extensible Exchange Protocol or BEEP)
- **Messaging Protocol**: Encoding information in a common XML format for understanding at both end-points of the communication (eg: XML-RPC, WS-Addressing, SOAP)
- **Description Protocol**: Public interface description for a web service (WSDL)
- **Discovery Protocol**: Centralized registry for web services to publish their location and description, as well as for clients to discover available services (UDDI not yet widey adopted)

8.3.2 SOAP

- Simple Object Access Protocol (SOAP) provides a standard packaging structure for transmission of XML documents over HTTP, SMTP or FTP. It allows interoperability between different middleware systems.
- Root element of SOAP message is called the **envelope**, which contains a:
  1. *Header*: Authentication credentials, and routing info/transaction management/message parsing instructions
  2. *Body*: payload of the message

8.3.3 WSDL

- Web Services Description Language gives description of interface for web services (in terms of possible operations)
- Standardized representation of input, output parameters, protocol bindings.
- Allows heterogeneous clients to communicate with the web service in a standardized manner

8.3.4 UDDI

- Uniform Description, Discovery and Integration standard, a global registry for advertising and discovery of web services
- Search by name, ID, category or specification implemented

9 Models for inter-process communication

- Interaction between processes can be classified along two dimensions:
  - First dimension: one-to-one vs one-to-many
  - Second dimension: asynchronous vs synchronous
- The following are the types of one-to-one interaction:
  - Synchronous request/response: The client send a message and blocks while waiting for the reply from the service. This style results from tightly coupled service and client interaction.
  - Asynchronous request/response: Client sends a message and the service responds to this message asynchronously. The client does not block while waiting as there is no guarantee of the service sending the reply within some constrained time interval
  - One-way notification: Service client sends a request to a service but no reply is expected
- The following are the types of one-to-many interaction:
  - Pub-sub: A client publishes a message which is consumed by 0 or more interested services
  - Publish/async response: A client publishes a request message and then waits for a certain amount of time for responses from interested services
- Advantages of asynchronous messaging:
  - Reduced coupling
  - Multiple subscribers
  - Failure isolation: If consumer fails then sender can still send messages and consumer can read them once it is up again. In a synchronous service the downstream client must always be operational
  - Load leveling: A queue can act as a buffer to level the workload, so that receivers can process messages at their own rate
- Disadvantages of asynchronous messaging:
  - Tight integration with messaging infrastructure
  - High latency in case of high load (message queue overflows)
  - Handling complex scenarios like duplicate messages, and coordinating request-response pairs
  - Throughput reduction caused by enqueue and dequeue operations as well as locking mechanisms within a queue.

9.1 Message Queue

- A form of asynchronous communication used in serverless and microservice architectures. A message queue provides a lightweight buffer which temporarily stores messages, and endpoints for software components to connect to (to be able to send/recv messages).
- A message is pushed into the queue and stays there until it is processed and deleted.
- Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads.
- Producer adds messages to the queue, Consumer reads messages from the queue and processes them.
- A single queue can be used by multiple producer-consumer pairs but only one consumer can read a message. Hence this model is used for point-to-point communication.

9.2 Pub-Sub Model

- The following are the components of a pub-sub communication model:
  - **Topic**: intermediary channel that maintains a list of subscribers to send a message
  - **Message**: Serialized messages sent by a topic by publishers
  - **Publishers**: Service that publishes the messages
  - **Subscribers**: A service that subscribes to a topic in order to receive messages published on that topic
- Advantages of pub-sub model:
  - Loose coupling as publishers and subscribers are not aware of one another and are independent of each other's failures. Hence independent scaling of subscribers and publishers is also allowed.
  - Scalability due to parallel operations, message caching, tree-based routing, and multiple other features built into the pub/sub model
  - Allow instantaneous push-based delivery hence removing the need for polling, hence causing faster response time and reduces delivery latency
  - Dynamic targeting as subscribers can dynamically add and remove themselves from a topic, and the topic server can adjust to changing numbers of subscribers.
  - Fewer callbacks and simpler code for communication makes it easier to maintain and extend.

9.3 REDIS

- Remote Dictionary Server, a fast, open-source, in-memory key-value data store
- It can be used as a database, a message-broker or a queue.
- In-memory storage allows for reduced seek time and allows microsecond delays in data access.

10 Monolithic and Micro-Services Applications

- Monolithic applications are built as a single unit, deployed on a single machine, and consists of a client-side application, a server-side application and a database.
- Microservice applications divide each component of the application into independent units that implement different parts of the business logic.
- Advantages of monolithic application:
  - Easy to develop
  - Simple testing and simple test automation
  - Easy to deploy
- Advantages of microservice applications:
  - Flexibility in adopting new technologies, maintaining smaller code bases.
  - Reliability, as one service failing doesn't affect the remaining ones.
  - Fast development due to reduced code base size, hence also easier to improve code quality.
  - Building complex applications is easier once the boundaries for components are decided, each can be developed independently and in parallel.
  - Highly scalable
  - Continuous deployment, meaning that microservice components can be independently updated without affecting the rest of the software

10.1 Service Oriented Architecture

- SOA breaks up the components required for applications into separate service modules that communicate with one another to meet specific business objectives.
- Microservice architecture is generally considered an evolution of SOA as its services are more fine-grained, and function independently of each other

10.2 Migration from monolithic to microservice model

- Some architectural challenges involved in this migration are:
  - Decomposition of monolithic software into independent components
  - Database decomposition in a consistent manner
  - Transaction boundaries
  - Performance and testing
  - Inter-service communication

10.2.1 Service Decomposition

- Don't add new features, start with existing loosely coupled components and identify those components which are ripe for enhancement
- Service decomposition leads to management and infrastructure overheads which can be resolved using containerization technologies to simplify deployment and configuration vastly

10.2.2 Database decomposition

- In a monolithic application, modules access data belonging to other modules using table joins. In a microservice application this can be avoided by using APIs to access data, or using projection/replication of data.
- Shared database tables, as well as current state information that are used by multiple components of a monolithic application can be modelled as a separate independent service

10.2.3 Transaction Boundaries

- ACID properties of a single database are easier to maintain than that of distributed databases in a microservice application.
- In a 2-phase commit, the controlling node first asks all the participating nodes whether they are ready to transact. Only if all nodes respond with a yes, then the controller asks them to commit. If a single node responds with no, then all nodes are made to roll back the transaction
- In a compensating or Saga transaction, each service performs its own transaction and publishes an event. The other services listen to that event and perform the next local transaction. If one transaction fails for some reason, then the saga also executes compensating transactions to undo the impact of the preceding transactions.

10.2.4 Performance and Testing

- Increase in resource usage causes microservice applications to perform slower
- This can be overcome by provisioning more hardware, logging to analyze bottlenecks, throttling, dedicated thread pools, and asynchronous features to improve performance
- Writing integration test cases is challenging as it requires knowledge of all microservice components and since such apps are asynchronous
- Solution is adopting various testing methodologies and tools and leveraging continuous integration capabilities through automation and standard agile methodologies