Module	Assessment Number	Assessment Type
Distributed and Cloud Systems Programming	A1	Individual Report

Coursework Report

Student Id : 2049867

Student Name : Roshan Parajuli

Section : L5CG3

Module Leader : Rupak Koirala

Lecturer : Saroj Sharma

Submitted on : 2021/05/09

Acknowledgment

I would like to express my deepest gratitude to my Distributed and Cloud Systems Programming lecturer Saroj Sharma and module leader Rupak Koirala for giving me this opportunity to research the cloud and distributed system with their architecture. Their guidance and continuous motivation helped me a lot in completing this report. I have taken efforts in this project. However, it would not have been possible alone. I would like to thank all the teachers and my colleagues for setting up the competitive environment for research and knowledge gain.

I would like to extend my thankfulness to the University of Wolverhampton as well as Herald College for providing the module contents and good cooperation in the ongoing semester and for providing a great topic for research which helped me learn a lot about the topic.

Table of Contents

Q. No. 1.

Application areas for cloud computing in FoundlingTech.com

In the retail industry, Cloud computing is becoming increasingly popular because of its huge use case scenarios. Cloud migration is a wise move for retailers aiming to not only draw new customers but also maintain the existing ones. For a huge company like FoundlingTech.com, and the involvement in the services covering the company website, CRM and the software development, the shift to the cloud computing would both be beneficial for the organization as well as the user base. Cloud computing offers retailers opportunities that both help to increase and maintain customer demands.

As a retail firm, it should become more flexible and open to changes while the exact opposite is the case. The in-house system has become a legacy system in the present context because of the difficulty in integrating with new technologies and being costly to run. Cloud system would help the firm deliver the smooth customer experience that the customers demand. It would also be better for security reasons as the data stored in the cloud is generally safer than the data store in the in-house servers. As a retail firm, the most important part of the job is conducting a successful business and not managing and troublesome hardware infrastructures. It would shift the company away from the goal. For such scenario, cloud computing would contribute to a great extent. In such way, the profits could be maximized and the unnecessary deviation of a retail firm to the chores that should not be much of a concern could easily be minimized.

As the firm provides the services for the creation of websites and software development, the firm could utilize the serverless backend feature from the cloud. The inclusion of the backend as a service and the function as a service in the cloud would help the developers get less burdened as the features like data and authentication can easily be handled in the cloud without the time consuming and costly provisioning and maintenance of the servers. It would result in the overall increase in the productivity of the employees of the firm as they would then be able to focus on the task that they are best at i.e., development with hassle-free deployment. As a retail firm with the CRM (Customer Relationship Management) service in-hand, it is a must to take no customers for granted as in today’s situation, the business environment has widely changed from what it was in the past. The customers today are more demanding and more informed, and a greater focus should be placed on the value as well as the quality/efficiency of the services provided. Not only the employee satisfaction, the fast and reliable servers in the cloud with the high degree of availability and scalability would also result in the increment in the customer satisfaction help the business grow the Customer Relationship Management service as well.

Advantages of moving the services to the cloud:

1. Cost savings: As the firm does not need to invest on the hardware resources and its maintenance cost, the capital cost would significantly be saved. The hardware is bought and maintained by the cloud service provider and need not be the concern of the customers as well as the firm.

2. Speed: The websites as well as the applications can be very quickly deployed in the matter of a few clicks. The resources needed for the deployment is automatically allocated within a few minutes.

3. No single point of failure: Fault prevention is a very important part that needs to be taken care of. When a component fails in a system, the cloud service makes it easier for other parts to take over in a matter of few seconds (mostly even less).

4. Burst ability: When a service requires an extra capacity but for a short period of time, the cloud would easily provide it while it would be hectic for the in-house services.

5. Scalability: The services in the cloud can be easily scaled up and down in a very short interval whilst in the in-house services, the services would mostly be over capacity or under capacity and only a few would be on the ideal capacity.

Disadvantages of moving the services to the cloud:

Although the advantages of the cloud outgrows the disadvantages it can not be neglected as well.

1. Initial migration problems may arise: Since the firm already has the services set-up in the way that would be complementary to the in-house system, the applications and the websites must be recoded to work in a new environment.

2. Risk out of hands: Although, the risk of data loss and theft is very low in the cloud, its not zero. The in-house private server is less likely to be the target of the hackers whereas the public cloud providers would face the attacks more. Risking a problem that is out-of-hands can be a more troublesome issue.

3. Lack of support: Cloud computing companies want their users to depend on the online help as well as the FAQs for the support which can be a very hectic job if the required resource for support is not found.

4. Risk of new government regulations: If a specific law is passed which prohibits a business firm to store data in any of the international cloud platform, the firm would have to reconsider the in-house servers again and would have to settle for it once and for all.

5. Forcibly agree to the policy: If a firm is dependent on a cloud computing service, it would have to follow the terms and conditions and play according to the rule of the service provider.

Potential for new services and business

Cloud provides a variety of services that can be integrated into the firm apart from the ones that are already there. The in-house infrastructure would limit a business to open for a new set of services because of the difficulty in integrating with new technologies in the legacy infrastructure. After the integration, the firm can analyze the needs and demands of the customers and opt-in for the various services that would be able to extend the business. The firm would be able to provide services related to IaaS (Infrastructure as a Service). With its usage, the firm could easily provide the virtualized computing resources to the users over the internet. With IaaS, the firm could provide services like high performance computing, big data analysis, storage, backup & recovery, as well as the environment for test and development. Another service could be related to PaaS (Platform as a Service) and with its usage, the firm could provide services related to the analytics or business intelligence and other services that enhance the applications such as workflow, directory, security as well as scheduling. Furthermore, the firm could also benefit from a lot of other services related to the SaaS (Software as a Service) which it has been providing till with the in-house IT infrastructure.

Q. No. 2.

Summary of “A brief introduction to distributed systems”

The trend of computers being portable and affordable started from the mid-1980s due to the development of powerful microprocessors as well as the invention of high-speed computer networks. The multicore CPUs that are extensively used now-a-days face the problem of adaptation and development of the programs for exploiting parallelism which gave rise to the concept of a distributed system. A distributed system is a system of computing where several different components located in different locations or different machines that communicate with each other and coordinate actions in such a way that it appears as a single unified system to the end-users. The computers in the distributed system are generally geographically dispersed which led to its naming. The computing element that is involved in a distributed system can either be a hardware device or a software process.

The **most important feature** of a distributed system is the collaboration, where the autonomous entities should act independently from each other while working together as a single unified system and the combination where entities or nodes should work together as a single coherent system. For a distributed system to be coherent, the collection of nodes should operate the same without the dependency of when and how the interaction between user and system is taken place. Achieving a coherent system can be difficult and needs to be dealt with as the distributed system consists of multiple nodes networked with each other and it is inevitable that a part of a system fails. The distributed applications are often make use of a separate layer placed logically on top of the operating system of the system called middleware. The primary purpose of the middleware is to extend over multiple set of machines while offering the same interface for each application in a distributed environment. The middleware services include communication service (e.g.: Remote Procedure Call) which permits an application to invoke a function that is implemented as well as executed on a remote server as it were locally available. Another service is the transaction service which offers special support for services distributed among several nodes in an all-or-nothing fashion. Service composition is the way of developing new applications by adding the existing programs together and reliability are also the services of the middleware. It also hides differences in technology, structure as well as behavior.

The **design goals** of a distributed system should be to facilitate resource sharing, to promote transparency, to hide the distribution of resources when it is excessive and to foster scalability. Resources are everything that can be linked to a computer system including printers, distributed sensors storage devices, etc. Such resources should be shared as it is efficient cost wise to have a single high-end storage which is reliable be shared rather than buying and maintaining different storage for different users. **Transparency** is hiding irrelevant properties of the system's components and structure. Distributed system should not overwhelm users with information that is non-relevant information from the user's viewpoint. Several different types of transparency exist depending on the property that is hidden from the users. Access transparency is hiding differences in data representation, component structure, protocols and how a particular resource is accessed. Homogenous access should be provided to the data, components as well as resources hiding the heterogeneity. Location transparency is hiding where a resource is physically located. The example of the location transparency is the

Universal Resource Locator (URL) which provides no information about the location of the physical distribution whatsoever. Migration transparency is hiding that a resource may have been moved to another location. The location of resources might change within a distributed system. It should be allowed without losing functionality as well as coherence. Relocation transparency is hiding that the resource may have been moved to another location while in use. It can be regarded as the special version of the migration transparency. Replication transparency is hiding that a resource is replicated. Replication helps users to provide faster accessible data for efficiency and in promoting tolerance to failures. Concurrency transparency is hiding that a resource might have been shared to the multiple users. Users should be able to work hassle-free with the distributed resources in a concurrent way. A distributed system should also take care of consistency i.e., ensuring resource safety even under concurrent access. Finally, Failure transparency is hiding the failure and recovery of a resource. Although, masking failures is one of the toughest issues in distributed systems, systems should exploit distribution to reduce the impact of partial failure into the overall system while hiding failures to users as often as possible. There are situations in which hiding distribution is not a good idea. As the notion of location and context awareness is important, it might also be best to expose distribution. Performance and comprehensibility should be considered while aiming for the distribution transparency. **Openness** is a major goal of the distributed system in which the system offers such components that can easily be integrated into other systems. The main issues for open systems are the interoperability, portability and extensibility. **Scalability** is another goal of the design system. The system is prone to scale up when the number of users as well as resources grows, geographical distribution of users and resources extends as well as when administrative domains increase. Lack of scalability causes issues where the computational capacity, storage capacity could be limited by the CPUs and disks as well as other network issues. Decentralized algorithms are best effective on the distributive systems as the failure of one machine would not ruin the whole algorithm and no implicit assumption about a global clock is required. Scalability faces certain trouble with communication as the communication in WAN is not typically synchronous as the communication in LAN and unreliable. Centralization being a mess is a shared trouble with size scalability. The three basic techniques for geographical scalability are: Hiding communication latencies, Distribution and Replication. In hiding communication latencies, the basic idea is to try to avoid waiting for responses to remote-service requests as much as possible. Asynchronous communications should handle requests whenever possible. When asynchronous communication is not feasible, alternative techniques like shipping code are needed. In distribution, the basic idea is to take a component, split it into parts and spread it across the system. The Domain Name System (DNS) is an example of distribution as it is hierarchically organized into a tree of domains, divided into zones and each zones are handled by a single server. In replication, the components across a distributed system are replicated in multiple locations for the purpose of increasing availability and solving the problems of latency. Caching is a special form of replication, but it is a decision by the client using a resource as opposed to replication which is a decision of the owner of the resource. When the data is stored into multiple locations, the problem with consistency is destined to become a bottleneck. As replication requires a global synchronization mechanism, it is very difficult to implement in a scalable way.

In administrative scalability, the trouble to be faced are the non-technical ones including policy of organization and human collaboration. As the components in the distributed system are dispersed along the network, not taking the dispersion into account during design time would end up making the system complex and riddled with flaws to be patched for later.

Distributed systems can be categorized into three classes: Distributed Computing Systems, Distributed Information Systems and Distributed Pervasive Systems. The key feature of **distributed computing system** is performing high-performance tasks using a large set of distributed computers. It is again divided into two classes: **Cluster and Grid computing systems**. The basic idea of cluster computing systems is to interconnect hosts in the same area through LAN where the hosts run the same OS and have the same architecture as other hosts. The cluster computer systems are homogeneous. As opposed to this, Grid computing system are mostly heterogenous as the resources from different organizations are brought together for the purpose of collaboration. The collaboration is built in terms of virtual organization. The architecture of Grid computing system contains several layers including fabric layer to act as a interface to local resources at a specific site, connectivity layer for the communication protocols and security protocols for grid transactions, resource layer for access control and the management of single resources, collective layer for handling access to multiple resources and application layer for applications operating in the virtual organization. Grid middleware later is represented by connectivity, resource and collective layers. Distributed information system involves itself with a set of distributed transactions i.e., operations on databases. The transactions are atomic, consistent, isolated and durable. A transaction can be made up of number of sub transactions which is called a nested transaction. Transaction processing (TP) monitor allows an application to access multiple server/databases by offering a transactional programming model with transactional semantics. A distributed pervasive structure is a part of our environment that is not regulated by humans. It should embrace contextual changes, encourage ad hoc composition as well as recognize sharing as the default. It can be used in home systems as such system should be self-configuring and self-maintaining. Sensor networks are a key component of pervasive systems. Sensor networks need to operate on energy friendly manner and the nodes of the systems are to be active only part of the time that they are being used.

To sum up, the prospects and benefits of distributed systems are huge, and it has several complications as well as opportunities. As systems are complex, distributed systems can help in the abstraction of complexity and improvement in the understanding and ease of use. A new set of computational systems are introduced through the distributed system which when ignored, lead to severe problems. There are several different types of distributed systems, depending on the context in which they are created, the goals they must accomplish, and the level of technology available.

Q. No. 3.

3. A.

Answer: The failed Lambda functions have been running for over 15 minutes and reached the maximum execution time.

As the question mentioned, the function would run 15 minutes on an average which implies that at some instances, it could run more than 15 minutes. But AWS Lambda terminates the function once it reaches the 15 minutes mark as it is the maximum execution time of a lambda function (Previously, it was 5 minutes which was then upgraded to 15), making it the most likely cause of the issue.

If the Lambda function with the average execution time of 15 minutes contained the recursive code, the total execution time would exceed the maximum execution time of the lambda function which would result in the failure of all the function invocations and not a few of them throughout the day.

The concurrent execution time limit error is not likely to be the error cause because AWS Lambda's default concurrent execution limit is 1000 which means that at any given time, 1000 concurrent executions of the function code is possible. If the execution exceeded the limit, it would result in all the functions piling up and not executing and not the few terminations throughout the day.

The ServiceException error is also not likely to be the cause because the internal errors on the AWS Lambda Service would result in all the lambda functions not running and not a particular one. Also, it is a rare case for a huge service like lambda to face the internal error several times a day.

3. B.

Answer: Dynamo DB.

DynamoDB is a fully managed, multi-region and durable database which can handle more than 10 trillion requests per day. It is also easily scalable with built-in security, backup and restore and in-memory caching features which would make it a perfect fit for the mentioned company as it satisfies all the needs mentioned. Even if the user base could be over 10 million, the durability, persistency, scalability and availability of DynamoDB would be able to handle it with ease and no downtime.

Amazon Kinesis is aimed for the real-time processing and streaming of large amount of data (gigabytes) per second. It is not used for the purposes of data storage but for the purpose of collection and processing of streaming data as well as for the analyzation purposes.

RDS, even though is a relational database service, lags behind the DynamoDB in the given context because of its difficulty in scaling due to the need of change in the instance size while scaling and the need of the usage of Multi-AZ instance for the increased rate in availability.

Redshift is the analytics database and a fully managed data warehousing service in the cloud. Since it is an analytics database, its primary purpose is for the analytics on the data and not for the usage in the transactional databases due to which it would also be a bad pick for the usage in the mobile application.

3. C.

Answer: AWS Elastic Beanstalk.

AWS Elastic Beanstalk is a service that is used for the deployment and the scaling of web applications with the features ranging from capacity provisioning, load balancing, auto-scaling to health monitoring. If the solutions architect was assigned to deploy without worrying the overlying infrastructure, Elastic Beanstalk would be the go-to option. AWS CodeCommit is a source control service that hosts Git-based repositories in the cloud. AWS CloudFormation is a service giving businesses and developers a simple way to model and set up Amazon Web Services by creating a collection of AWS and third-party resources. Amazon CloudFront is a content delivery network (CDN) service to deliver data and APIs with very fast transfer speeds and very less latency. Since, the other services do not fulfill the need of the solutions architect, Elastic Beanstalk is the only one to choose.

Q. No. 4.

4. A.

a.

The statement might be false because the application and reachability of cloud computing is very huge and not limited to the usage in special shopping days. When cloud computing was invented around 1960s, even though the retails experienced huge traffic in a single day, it may not have been worth it to shift the whole infrastructure in the cloud just for the scalability reasons for a single day as it would require a lot more work to shift the infrastructure to the cloud than to handle a huge burst if traffic for a single day. The online shopping days would be a very narrow field of operation of cloud computing.

b.

The statement might also be true because of the elasticity that the cloud provides. Special shopping days would require massive infrastructure because of high traffic but would last for a small interval. It would not make sense for a company to build such great infrastructures just to handle the demand for a short interval time and the infrastructures would remain unutilized at the other point in time. To tackle this problem, cloud computing could contribute greatly because of its scalability. The companies could increase the scale of the servers only at the time of need and decrease the scale as per the need.

4. B.

Even though the mobile devices have many cores and powerful GPUs, there are still some reasons due to which offloading application functionality from the smartphone would be a better option. First is the **battery capacity** of the smartphone. The average battery capacity of a normal smartphone is around 2500 mAh which is insufficient for most actions. Another reason would be the lack of sustainability in a mobile device due to the **thermal condition** as mobile devices get easily heated up and even if it performs greatly for some interval of time, the efficiency is likely to decrease. Mobile devices are also very **prone to ransomware and malware** with very little to no backup security against various virus attacks.

Q. No. 5.

Architecture and internals of job execution in Spark

Spark is an open-source engine for processing large-scale data in real-time across various clusters of computers providing high-level Application Programming Interfaces (APIs) and packages in multiple languages including Java, Scala, Python as well as R. The APIs and packages can be utilized for graph processing, continuous streaming, machine learning to operate on large scale datasets and a lot of other applications. Due to its support for multiple programming languages, it is also called a polyglot.

Before getting into the internals of the working mechanism of Apache Spark, it is important to know about its predecessor i.e., Hadoop MapReduce. It is termed as a predecessor because it is rather old and is inefficient in terms of fast processing of data or with a continuous stream of data. In distributed computing, the main part of the process is the effective division of tasks across multiple hosts (in this case, machines). The proper division of tasks across different machines would result in the effective utilization of those machines. For the same purpose, a programming model was developed whose job was to split tasks across multiple machines, which is known as MapReduce. It is the way of structuring the whole computation so that it would be easier to organize and run in multiple machines parallelly. It works on the principle of map, shuffle, and reduce. The main drawback of MapReduce was it allowed the user to only write batch processing jobs and was slow. Spark was then introduced to be able to process a continuous stream of data in memory, which would result in faster processing in comparison to Hadoop. The gap was so high that the in-memory processing used by Spark turned out to be almost 50 to100 faster than that of Hadoop with MapReduce.

Spark works on the principle of having a master coordinator, namely known as the driver, and many workers, also known as executors, which are distributed through the help of master-worker architecture. Each component of the architecture run in their own Java processes. Spark can work with different clustering technologies e.g.: Mesos, Yarn as well as standalone. The architecture consists of a master node that contains a driver program that internally uses spark context, the entry point of application. Spark context can be regarded as the gateway to all the existing Spark functionalities. The driver program interacts with the cluster manager by taking the application requests to it. Spark supports a bunch of cluster managers for the job. The built-in cluster manager is the Standalone Cluster Master, but it can also be configured to use any other cluster managers including Yarn by Hadoop and Mesos by apache, etc. The job execution part of a particular cluster is well taken care of by the driver program and the spark context. The combination of both separates a specific job into the worker (or slave) nodes. Whenever a transformation is done to create the Resilient Distributed Dataset (RDD), a fundamental data structure used in Spark, it is then distributed across multiple nodes for the processing of data.

Launching a program in Spark is a fairly simple job. The spark-submit is a script that is used to launch applications on a cluster by using any of the supported cluster managers through an interface that is uniform so that each application need not be configured manually. There are multiple options for the spark-submit to connect to the supported cluster managers and moderate the number of resources the application gets which results in its flexibility. The starting point of an application i.e., the main() method is invoked in the spark-submit which then launches the driver program. The driver program creates RDDs, performs transformations as well as actions, and creates spark context. When a Spark shell, a command-line environment with the features such as auto-completion written in Scala, is launched, the driver program is created then and there. Once the driver terminates itself, the application is finished and exits. The driver then creates a graph called the operator graph and when an action (e.g.: reduce, first, count, etc.) is triggered, the operator graph is submitted to the Directed Acyclic Graph (DAG) Scheduler. In short, DAG is a combination of Vertices as well as Edges where vertices and edges signify two different things. The RDDs and the operations that are to be applied to the RDDs are represented by the vertices and edges, respectively. The main task of the DAG Scheduler is to divide the operator graph that is provided to it into the stages of the tasks and submit every task to the task scheduler. The task scheduler is then ready to launch the provided tasks with the means of cluster manager. DAG also helps in achieving fault tolerance. The driver manager then requests the chosen cluster manager for the resources that need to be allocated to launch the worker nodes (executors). If everything goes well, the cluster manager then launches the worker nodes with the required resources. After that, the driver sends the whole work to the allocated worker nodes in the form of tasks. A particular task is a unit of work that is to be sent to the worker nodes. After the worker nodes successfully process the tasks, it sends the produced result to the driver program with the help of the cluster manager. If a spark executor fails in the middle of the application, it is handled elegantly by the DAG Scheduler as well as the cluster manager. If a particular executor fails, the cluster manager assigns another node to continue the processing work of the dead executor with the help of the DAG Scheduler.

B. Why Spark is considered “in-memory” compared to Hadoop?

Hadoop uses a programming model known as “MapReduce” which breaks the tasks into three steps: Map, Shuffle and Reduce. The problem with the MapReduce was that while performing each operation, the input was fetched from the storage and after the computation was done, the output was again written to the storage which would result in a slow performance because of the continuous read and write operation from the storage. In contrast to Hadoop, Spark does the same task in a much more efficient way by reducing the disk reading and writing operations. In Spark, the data needed for processing is stored in Random Access Memory (RAM) and processed parallelly. Spark uses a general engine supporting cyclic data flows and tries its best to keep the data in memory during the completion of a job. The data can be easily modified without the process of serialization (a process of converting an object into the format for the storage). Even though Spark is considered “in-memory”, it does store the data into the disk if it is not able to hold an RDD in memory between the steps. Due to the in-memory nature of Spark, it does invite the complications like the out of memory problems which is mostly not a case in terms of Hadoop as it would just write everything out directly to the disk.

C. Resilient Distributed Dataset and its operations.

Resilient Distributed Dataset (RDD) is the collection of elements that are shared and computed across the different nodes of the cluster and supports operation in parallel. They are the fundamental data structure of Apache Spark supporting in-memory processing naturally and contain the property of immutability. RDDs are simply an arbitrary collection of objects (Java or Scala) which represents data.   RDDs are resilient in the sense that they are heavily fault tolerant, partitioned in the sense that it is broken up into smaller chunks called partitions, distributed in the sense that it is spread across the cluster rather than on a single machine and immutable in the sense that once it is defined, it cannot be changed but can be transformed into another RDD with the help of certain transformation. It is physically stored in the JVM driver as an object and refers to the data which can either be residing in the permanent storage like HDFS, HBase, etc. or in a cache like memory, disk, or memory as well as disk or even on another RDD. If a particular RDD is used several times, caching it would be very beneficial in terms of performance. It is fault tolerant because even if it fails, it can easily be recreated as each RDD knows how it was created.

RDD supports two types of operations:

i. Transformation
ii. Action

All the work is performed in the clusters in the sequence of the transformations and actions.

Transformation is a function which yields a new RDD from the existing RDDs by transforming the elements of the given data. Certain operations like map, filter, join, union, etc. transform a given set of RDDs into a new one. Transformations are lazy because the data is not loaded in the RDDs as soon as it is transformed. Spark keeps a record of the transformations in the

DAG (Directed Acyclic Graph) and the operations are executed only when a certain action is triggered on it. Two types of transformations can be applied to RDDs: Narrow and Wide transformations. The main difference between them is that the narrow transformation does not require reorganizing of data between partitions while the case is opposite in case of wide transformations.

Action is an RDD operations that return non-RDD values as opposed to the transformation. Certain operations like reduce, first, count, collect, saveAsTextFile etc. are the common actions which returns a value after running computation on RDD. Actions invoke the execution from the first to the last RDD. The values produced by an action is either stored in the external storage system or in the drivers. Since a transformation is lazy in the sense that they do not get executed right away and wait for the action functions for the execution, the actions upon being called on the transformed RDD will compute the result and bring its laziness to the spotlight. The output of Transformation is in a sense, an input to the actions. To sum up, Actions uses the lineage graph to load all the data into original RDD, carry out all the transformations and return the outcome to either the driver program or write out into the file system.