Advanced Database Systems Midterm Fall 2022
Professor Joann Ordille

This midterm is a take-home, open-book exam. You may use class notes, recordings, readings and exercises as well as your own independent research to answer these questions. You **CANNOT** copy verbatim from anyone without attribution. Such copying is plagiarism and is not allowed. Your work must be your own. Your submission for the midterm will be analyzed for similarity to other submissions received as well as other available sources. Students copying material, even with paraphrasing, from each other will have their grades penalized, and will likely be reported to the academic integrity office. For more information on Rutgers policies on academic integrity, see the syllabus. A good way to avoid problems is to **NEVER** write your answers with others or to look at the written answers of others.

There are four questions on this exam, worth 25 points each. Each question will be graded on: (1) completeness, (2) clarity, (3) accuracy, and (4) precision.

1. A successful retail company has several stores in the Northeastern United States. The company's sales show seasonal behavior with peaks in December and August. The company has an internal computer network which connects all the stores and supplies its computing needs including communications, inventory control, pricing, supply chain management, distribution and human resource management. The company plans to begin selling its products online throughout the United States and Canada. It hires you as a consultant to plan how it will use technology to implement and manage its new Internet store, and how that technology will be integrated with its existing systems. Describe your recommendations to the company, and the potential benefits and risks of your plan.

Key word: 1. how big the company is 2. since it wants to move the selling part only it needs server (do u want part online or move everything online) (pros low latency, fast to deploy, low cost may be, cons u cannot control the server level, could be potential fail down, extra IT staff) 3. If it is part only, how do I merge the existing hardware to new ones?

The first thing I think we need to identify is how large is the company, whether it is a large retail company like amazon or it is relatively small. Secondly, how should we utilize the existing hardware.

My suggestion here is that we should use Cloud based schema for selling its products online, here is the reason why.

1. It provided low latency for users from different regions and even different nations. The retail company is based in the Northeastern part of the US which means its original targeting group it is just local citizens who just live there. But its customers will be using their online ordering service from multiple different states and even different countries after they begin selling products online. This makes latency become the No.1 priority among their customers' experience since that no one wants to wait after clicking the button and wait dozens of sections to get the resources they require from the server. Using the Cloud service enables you to set up multiple servers across the US to lower that latency, which gives customers better experience.
2. Moving the online sell part will lower the potential cost and enable auto-scale. As we all know, the server is really expensive, small companies may even be able to afford to invest that kind of money in a department which has not even started making money. Not to mention that you need to rent a building/room to hold that infrastructure. Also, it is truly hard to provision the development of the online selling department. Even though you can argue that you have hired analysts to dig information from existing selling data, it is unlikely that they will find a patten from the selling data generated by offline selling. The success of online selling is driven by several factors like user preference, shopping experience, brand attachments, etc. So, you will be unlikely to know how many

infrastructures you need to satisfy the requirements. Instead of buying hardware and building a server from the ground, you just simply rent the service from cloud computing services companies also known as IaaS. This allows you to have a slice taste of online selling without investing too much money at the beginning. Not to mention that your company's selling records show a pattern of seasonality, which indicates that you maybe need to purchase that kind of amount of hardware to meet the workload in order not to lose any business. On the other hand, the servers you just purchased are sitting there not producing any benefit since you don't have the same burden of computing resources over most time of the year. But you could just utilize the auto-scale option to let the service provider add more servers when experiencing peaks of service like August and December or due to expanding business. You don't need to repurchase infrastructure and do the whole thing again as the start of the online selling business.

3. Less server level control and indeterminate contingencies. When utilizing the Cloud server, you only have to do CREATE, DELEATE, UPDATE and Insert query via an API. This should not be a problem most of the time. But in some cases, the performance of the server may not reach the theoretical optimal performance since you only have an Application-Level of control which you only allowed to tune the server on top of something someone build. This means that once you experience such problems the only option you have is to ask a certificated representative for that service to look into that problem. It not only takes money to fix the problem but also potential loss of business during the maintenance of the server. Additionally, there also could be some accidences happen to the facilities where companies host their cloud services. On the one hand, that very location could experience hurricane, tsunami. It is very unlikely that it will happen

since the data centers were built on pre-selected locations which have mitigated environmental risk. Even if it happens, the Cloud service provider could route that client request to the nearest data center to exchange latency for service. On the other hand, the server could fail due to human error. This could be a potential risk.

Collaborate the existing hardware with the cloud. Moving the selling related database to the cloud only maintains the human resources on local servers. The reason why I suggest this is that it is relatively less often to change and less dependent on latency when compared to high frequency selling. As the selling part, my advice is to build a distributed database system on multiple servers across the nation and Canada. When handling online selling, the customer sends a request to the server which has the minimum distance to that location. The server can validate the order with inventory to see that the storage is still stocked once the requests reach. Then the server can put together the key information via the request just received to form an official order to acknowledge the shipping department. All retail department and shipping department should keep their address in the cloud table to let the manager have an idea of current distribution. We can also calculate the distance between known shipping or local facilities and the customer location to accelerate delivery or pickup. The data integrity is handled by the Distributed Database Manage System. You do not need to worry about overselling. On the other hand, offline selling inventory control is separated from online. Inventory controls are maintained completely locally and would not change any records of the database once their ordered stock arrives. They can manage their inventories by themselves, or your company could gather inventory daily to gain data and achieve supply chain control. Moreover, the company identifies that the local stores are in the area of

Northeast part of US so there is no need to move the Human Resource part to the cloud, otherwise it would be a waste of existing hardware.

2. ACID Transactions are the gold standard for consistent and correct databases for many applications. What are the A (atomicity) and the I (isolation) in a relational database transaction? Why are they useful features? Compare and contrast how atomicity and isolation are supported (or not) in relational databases systems, Bigtable and Amazon Dynamo.

1) What are the A (atomicity) and the I (isolation) in a relational database transaction?

Atomicity:

(1) Meaning: Atomicity means a thing that can not be subdivided which means that it has to either executed completely or roll back to its original status. Example: if you are making a transaction via bank. First, you withdraw 100$ and then deposit 30$. The database shuts down after you withdraw 100$. Now your friend is buying a 15$ product, this will cause logic error since you should have 30$ in your bank account but since the database shut down before executing deposit command your balance will be 0$ even you actually give back the 30$ to the bank. In order to achieve the database atomicity, you need to maintain 2 logs, undo log and redo log. The redo log is just the copy of every command you do on the database, the undo log is reverse command so that the database can undo the command after database fails. So, the rollback process is to use undo log to achieve the original status before you manipulate the database and use the redo log to let the database reach the status what it should be if no accident happened.

(2) Why useful? I personally think it maintains the consistency of the database. It only allows two states: “original” and “done”. If there is any state between these two states, it will create a gap between the general income and outcome of the database. For example, like the money transfer on top. The 30$(gap) is lost due to the “Middle” state generated by not performing the rollback after breaking the standard of atomicity.

Isolation:

(1) Meaning: Isolation usually defined in multiuser database to make sure that concurrent transactions do not interfere or affect each other when reading the same properties. Example: If you are depositing money to your bank account at T1, the original balance is zero. And your parents also send your tuition fees to your bank account at T2. Both of your parents and you execute read operation gain the same result of 0$. At T2, you successfully deposit 30$ and database commits that job at T4. Your parents, on the other hand, put 1000$ to your account at T3, database commit that job at T5. The logic balance should be 1030$ but due to not isolate your record, instead of being added by 1000$, will be only updated to 1000$ since the original read at T2 is already changed to 30$ at T4. To prevent this, we use lock and MVCC to satisfy different isolation levels.

(2) Why useful? To identify the degree to which concurrent transactions affect each other. In some contexts, the applications have different tolerances to the information visible to concurrent transactions. There are 4 different isolation levels in order to deal with 3 problems --- dirty read, non-repeatable read, phantom read. First, read uncommitted, the lowest level. In this level, a transaction can read uncommitted information from another transaction so all concurrent problems could happen. Second, read committed let the transaction read only after committed to deal with dirty read. Third, repeatable read forbite read while other transactions update the table to handle non-repeatable read. Fourth, serialization lets transaction execute linearly to solve phantom read.

2) Compare and contrast how atomicity and isolation are supported (or not) in relational databases systems, Bigtable and Amazon Dynamo.

(1) atomicity: Relational databases use redo logs and undo logs to achieve atomicity. In Amazon Dynamo, it has two phases to achieve atomicity, phase 1, creates a transaction record and index records. Also, save old values to snapshot table and update record, phase 2, if the result is committee successfully, the local transaction will atomically discard the change. Otherwise, it will atomically rollback the database based on the record we store locally then discard the record. In Bigtable, it has commit log for atomicity, all the modifications are stored to the commit log before being applied. This makes sure that when servers go down the server can based on the commit log to recover and re-apply the operation.

(2) Isolation: In relational database system, there are 4 different isolation levels in order to deal with 3 problems --- dirty read, non-repeatable read, problems --- dirty read, non-repeatable read, phantom read. First, read uncommitted, the lowest level. In this level, a transaction can read uncommitted information from another transaction so all concurrent problems could happen. Second, read committed let the transaction read only after committed to deal with dirty read. Third, repeatable read forbite read while other transactions update the table to handle non-repeatable read. Fourth, serialization lets transaction execute linearly to solve phantom read. In Dynamo, it gives users different options to handle different application requirements. Similar to relational databases systems, it gives two different levels, read committed and serializable. Read committed ensures that operation only returns committed values for an item. Serializable guarantees the results of concurrent operation are the same as if no operation begins until the previous one finished. In Bigtable, people use consistency model to achieve isolation. By default, replication for Bigtable is eventually consistent which means that it takes seconds

or minutes to synchronize among all replications. But if the application needs a high consistency level like read-your-write consistency you need to configure all cluster to be single-cluster routing. For strong consistency, it basically performs like a single cluster. The additional clusters are just for fail over.

（https://www.databricks.com/glossary/acid-transactions）

（https://en.wikipedia.org/wiki/ACID）

3. A DynamoDB database has sensor tables for a series of years starting in 2000, called sensor-yyyy. Each sensor table records temperature readings every few seconds from tens of thousands of ocean sensors and is similar to the AmazonDB sensor table in *Seven Databases in Seven Weeks*. DynamoDB also stores a location table that records the location of each sensor and its full set of capabilities including recording temperature, light level, currents, etc. Each sensor table is partitioned by sensor id and the location table is partitioned by location. A researcher wants to study a sample of the data by accessing the sensor tables for years 2000, 2005, 2010, 2015 and 2020. The researcher wants to use the location table to find the sensors in a geographic region, and then extract average temperature readings per day from the given years into a table for further analysis. The researcher plans to analyze several geographic regions to start, and ultimately to analyze all geographic regions. Explain a process and tools that the researcher can use to create the needed tables easily, and why you chose this solution.

Personally, I think it is more likely the architecture of DynamoDB part presented in the 7 databases in 7 days. Since the data is already gathered and stored within the DynamoDB, the only thing we need to do is extra, aggregate them and analyze them.

The ideal process is below, first extra the location table and sesor-yyyy table with right year via Data pipeline. The data pipeline also utilizes the EMR not only to boost the speed of transfer but also allow you to do data aggregation in this step. The third step is to store the data to S3 database in the format of JSON and then use Athena to query data with the target sensor location. The last step is using statistic to do analysis.

It is easy to extract the average of the data since we transfer the data stored in DynamoDB via data pipeline. EMR, also known as elastic map reduce, performs a task to group the data by days to get the average temperature from the JSON structured data where table name indicates the year meet the requirements and store them into AWS S3. Also, Apache DistCp, AWS also have their version bases on that to optimize the process input and output data from S, allows you to quickly gain the data from S3 to the EMR HDFS to do aggregation. This step reduces the cost by utilizing Hadoop to minimize time and space requirement for later steps. One sensor can produce millions of rows of data per year, by calculating its average temperature readings per day it reduces the records from millions to the level of tens of thousands. Then here is the real problem, how do we do join based on the location? Amazon DynamoDB is integrated with Apache Hive, a data warehousing application that runs on Amazon EMR. Hive can read and write data in DynamoDB tables, allowing you to perform join operations on DynamoDB tables by creating a Hive external table on location table and sensor table. After that use sensorID as FK to join. Finally, put the table to S3 in the format of JSON. Since it is a merged table, you can use location attributes to filter the geographic location you want to study.

（https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EMRforDynamoDB.Querying.html）

4. Define cloud native computing. Describe its advantages and disadvantages. Illustrate your discussion with examples.

The term cloud native computing refers to the approach in software development that utilizes cloud computing to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds.

Advantages:

1) Performance. It allows you to be able to access native cloud features of the public cloud to provide better performance not only you gain powerful computation power, but also the feature of native cloud enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil. For the physical server part, data centers across the world allow your customers to have low latency experience.
2) Cost. Someone could argue that we can set up different private servers near the hot spot of the business. But this will certainly being extra cost, since that you need to purchase equipment, rent houses, hire technicians, and get permission to permission to run business locally. Besides, you have to make a huge investment in hardware at the beginning of the business. The price won't be low since you have nothing to bargain with. Compared to the cloud native part, you can just simply rent a server (IaaS) and launch your business, there is no additional cost, you pay what you use. If the server fails, the cloud service provider has mechanisms and representatives to protect and recover the data.
3) Scalability. The requests you sent to the server can't be measured at the start of the business since it is very hard to predict how the business will develop. Failing to meet the

requirement of computing power and storage requirement is potential risk of losing business. If you overbuy the equipment, it is just an asset you cannot cash, either way, it costs company money. But cloud native allows you to add instances when necessary. There are cyclical events such as the COVID and Sars and it is impossible to predict these events. It is a waste when business hits the ground, but auto-scale could remove instances at no cost. It enables you to dynamically adjust the computation power based on your business requirement.

Disadvantages:

1) Low efficiency on maintenance. It is hard to quickly fix your service and put it back online when experiencing a complete failure of the cloud service since it takes time to let the representatives acknowledge what is happening and what causes the failure of the application. You usually don't have a server level control which slows down your repairment.
2) Security. Since you are moving your database online, your sensitive data may not always be safe with the cloud provider. For example, Facebook was breached before August 2019 and choose not to notify its users that their personal information was stolen – and shortly after that, posted on a public database. These could be a potential risk of your services and losing customers since no one feels safe using your services. And it is possible that the cloud service shuts down due to human error by the employee of cloud service provider and nature disasters. Instead of moving everything to the cloud, you basically want to run a synchronized version of database and host service on local to prevent these rare occasions happening.

3) Cost. The cost may not necessarily be low when using the cloud. Your maintenance could be higher since you are using the cloud representatives to do the repairment. Even if you have full control of the server, it generally takes more time to identify the problem because there are too many variables which could affect the performance or trigger the error.
4) Dependency. When it comes to cloud native, your application may need to be run in a certain environment with a handful of hardware, software or operating system. Plus, the newer technologies may even be vulnerable to attacks which brings another security issues.
5) https://github.com/cncf/toc/blob/main/DEFINITION.md