Washington State University
CPT_S 415 – Big Data
Online

Srinivasulu Badri

Assignment 4

Name: Nam Jun Lee

Student Number: 11606459

1. [Parallel Data Models]

a. Assume a program P running on a single-processor system takes time T to complete. 20% of P can only be executed sequentially on a single processor, and the rest is "embarrassingly parallel" in that it can be easily divided into smaller tasks executing concurrently across multiple processors. What are the best time costs to execute P using 2, 4, 8 machines (expressed by T). What are the speed-up respectively? What are the optimal speed-up given infinitely amount of machines?

Execution time of sequential S: 20/100 = 0.2

Execution time of embarrassingly parallel P: 80/100 = 0.8

Using Amdahl's law:

Time cost: $T_n = (1 - P) + \frac{P}{n}$; Speed-up: $S_n = \frac{1}{(1-P)+\left(\frac{P}{n}\right)}$

When 2 machines:

Time cost: $T_2 = (1 - 0.8) + \frac{0.8}{2} = 0.6$

Therefore Speed-up: $S_2 = \frac{1}{0.6} = 1.6777.. = 1.67$

When 4 machines:

Time cost: $T_4 = (1 - 0.8) + \frac{0.8}{4} = 0.4$

Therefore Speed-up: $S_4 = \frac{1}{0.4} = 2.5$

When 8 machines:

Time cost: $T_8 = (1 - 0.8) + \frac{0.8}{8} = 0.3$

Therefore Speed-up: $S_8 = \frac{1}{0.3} = 3.3333.. = 3.33.$

Also, find the optimal speed-up given an infinitely number of machines:

$T_\infty = execution\ time\ of\ sequential = 0.2$

Hence, Speed-up: $S_\infty = \frac{1}{0.2} = 5.$

b. Describe and compare the pros and cons of the three architectures for parallel systems.

There are three architectures for parallel systems:

1. Shared Memory

   Shared Memory is an architecture that enables program processes to exchange, read, and write data faster than regular operating system services. All processors also have parallel access to all parts of memory.

   (1) Pros:

   - Accessible to all through data in memory
   - Easy to program

   (2) Cons:

   - Non-expandable
   - Shared memory and network bottlenecks
   - Cache consistency issues when updating data
   - Expensive to build
   - Difficult to scaleup

2. Shared Disk

   Shared Disk is an architecture used for distributed computing where nodes share the same disk device, but each node has its own memory. In the event of a failure, there is an active node that shares memory, and all cluster nodes can access the disk.

   (1) Pros:

   - If a processor fails, the other can take over, since the database is resident on disk (Fault tolerance)
   - Better than shared memory (scalability)
   - All I/O to go through a single network

   (2) Cons:

   - Disk subsystem is a bottleneck
   - Not scalable beyond a couple of hundred processors

3. Shared Nothing

   Shared Nothing is an architecture used for distributed computing, where all nodes consist of processors, main memory, and disks, each node is independent and the other nodes are interconnected to the network.

(1) Pros:

- Easy to scaleup
- Communication costs and assess to non-local disks (Cheap to build)

(2) Cons:

- Hard to program

2. [ACID vs BASE] This set of questions are related to data consistency

a. What is CAP Theory? Consider an example cluster that contains three servers S1, S2 and S3, each located in a different geographical areas. Assuming data is partitioned across the three servers, explain CAP theory for the example cluster.

CAP theory is that database systems can meet only two of the three characteristics of consistency, availability, and Partition tolerance, and not all three.

1. Consistency

    - All replicas contain the same data version.
    - Clients always view the same data

This means that for all reads, nodes must always have the same data

2. Availability

    - System remains operational on failed node
    - All customers can read and write at all times

In other words, even if some nodes in the cluster fail, actions such as read and write must always be returned successfully

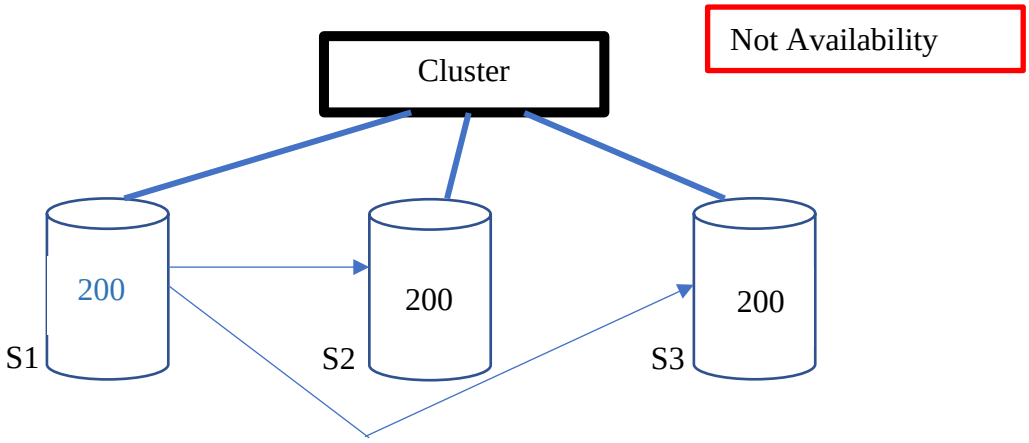
3. Partition Tolerance

    - Multiple entry points
    - System continues to operate when system is split (communication malfunction)
    - System functions properly on physical network partitions

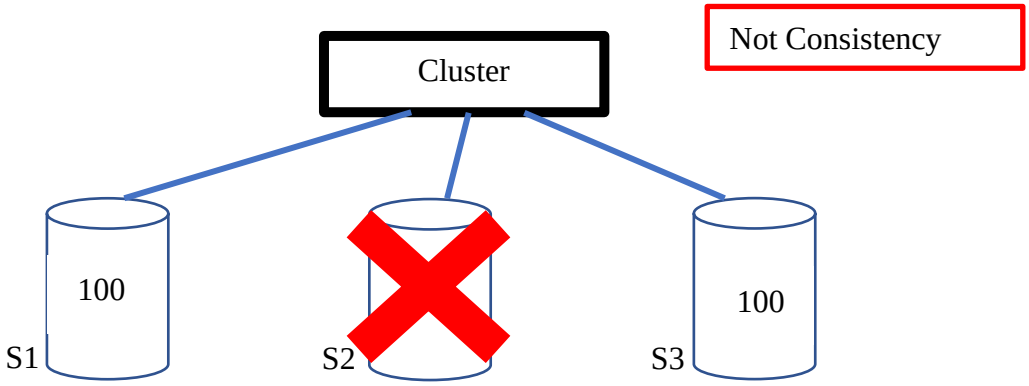
That is, it means that even if a communication failure between nodes occurs, it must operate normally.

Example cluster that contains three servers S1, S2 and S3:

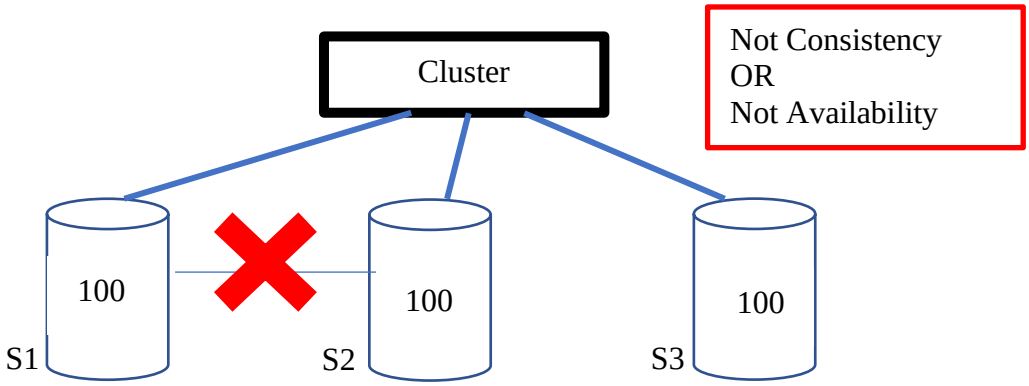
If added 100 more to S1, but it was not updated to S2 or S3. Then, S1 should be updated to S2 or S3 immediately. It can perform this update on a Cluster or S1. If so, the data remains consistent, but excessive real-life time leads to poor availability.



If the S2 server is broken, you can check the data with S1 and S3. It can be said that there is availability in this respect. However, this can be said to be inconsistent because when the S2 server is rejoined, the data of S1, and S3 are different from S2.



If the network connecting S1 and S2 fails, the two servers operate normally, but it is difficult to have the same data because there is no intersection between the servers. This means that it becomes inconsistent. If spend a lot of time waiting for the network to recover to maintain consistency, this reduces availability.



b. Consider the relation Accounts(acctNo, customerName, balance) with acctNo as primary key. Consider the following two SQL statements that conduct a request "transfer $200 from account A1 to B1".

   i. Add $200 to account B1:

      UPDATE Accounts SET balance=balance+200 WHERE acctNo = B1

   ii. Subtract $200 from account A1:

      UPDATE Accounts SET balance=balance-200 WHERE acctNo=A1

   Use this example and necessary scenarios to show when Atomicity, Consistency, Isolation and Durability can be violated.

1. Atomicity

If the first UPDATE statement is successful but the second UPDATE statement fails, the problem arises that the B1 account balance increases by $200, but the A1 account balance does not decrease by $200. It is atomicity that all work must either fail or return to success. Therefore, this violates atomicity because all operations in one transaction have not been successfully processed.

2. Consistency

If the first UPDATE statement is successful, but if A1 does not exist, the second UPDATE statement will fail. The balance of B1 increases by $200, but the A1 account does not exist, which violates the constraint of "acctNo as primary key". This violates consistency because the database becomes inconsistent after the transaction occurs.

3. Isolation

If two transactions run simultaneously and the balance of account B1 is zero, the same result as running each remittance operation in succession will not be achieved. Therefore, this violates isolation.

4. Durability

After the first UPDATE statement was successfully executed, a system error occurred while executing the second UPDATE statement and was terminated. There was no record of the transaction when re-connected. Durability should return to before a transaction is performed if it is terminated due to a system error before a single transaction is logged. Therefore, this violates durability.

c. What is “BASE”? give an example of “BASE” data consistency model and compare it to ACID.

BASE, it stands for Basic Available, Soft state, and Eventually Consistent, which is a database that provides flexibility and fluidity that is easy to manipulate data. This BASE database prioritizes availability and has the advantage of being able to manipulate data easily and quickly, while there are times when it lacks consistency.

Compare BASE data consistency model and ACID:

Consistency is important for the ACID data model, while availability is important for the BASE data consistency model. The BASE data consistency model is consistent but relatively less consistent than the ACID data model. Conversely, the BASE data consistency model is better than the ACID data model in terms of availability. In terms of tasks, the ACID model is complex, but for the BASE data consistency model, there is a flexible and fluid way to manipulate the data. In the case of the ACID data model, it is mainly used by financial institutions that value security and consistency, and in the case of the BASE data consistency model, it is mainly used by marketing or customer service institutions that need to analyze customer sentiment. Therefore, it is impossible to distinguish between better and worse data models. Because each has its own pros and cons.

3. [Quorum Consensus]

We introduced Quorum Consensus method to ensure consistent data can be fetched in read/write operations. Describe Quorum Consensus and explain why it works.

Quorum Consensus is one of the distributed lock manager-based concurrency control protocols in distributed database systems. This quorum improves through put() and get() task performance.

N is the size of replicas; W is writing; R is reading

This Quorum Consensus has rules: W+R > N

This rule prevents read-write impulses. This means that two transactions cannot be read and written at the same time.

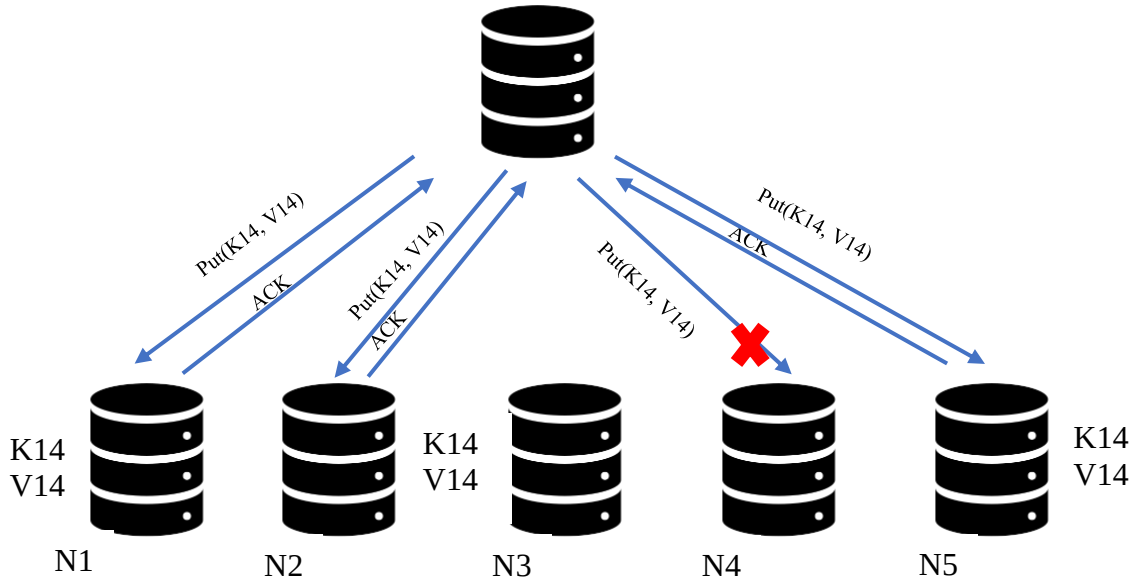
This Quorum Consensus aims to prevent partitions from producing inconsistent results, preventing updates from propagating successfully to subgroups of replicas and works by updating offline, although other replicas already have copies. It means that there is at least one node that contains the update.

For example:

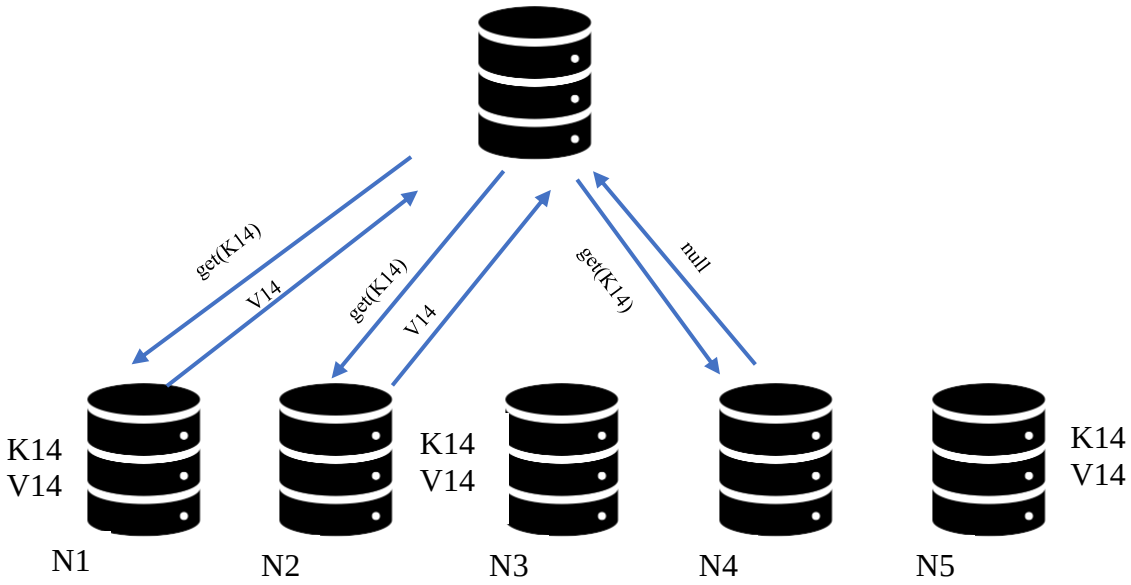
N = 4, W=3, R=3

Replica set for K14: {N1, N2, N4, N5}

Assume put(): N4 fails because W = 3



Assume get():



Therefore, we know that issuing get() to any three nodes out of four will return.

4. [Column Store]

Explain the major features of column stores in terms of data storage, supported operators and query processing.

Column stores has the function of sequentially storing all values of a column in a database page. This is to store data in a disk column by column. Also, reduces the data that needs to be taken from memory and can amortize the cost of bringing in many tuples.

Data storage:

The reason for using column storage can be much faster than row storage for some applications and can improve the cache effect because only the required columns can be imported. It also enables better compression than other storages in terms of data compression. However, the speed is slow for other applications, such as OLTP, which has many row

inserts. Therefore, column storage is suitable for large, read-intensive data stores that are read-oriented.

Supported operators:

Operators supporting 10 queries in the column stores mentioned in the lecture notes:

1. Select: Used to display table data, and you can filter the data by adding many clauses.

2. Project: Displays columns of relationships or tables based on the specified properties.

3. Join: Join the projection according to the predicate.

4. Aggregation: Performs calculations on a set of values and returns a single value. (Ex: Average, Count, etc.)

5. Sort: Sort all columns of projection.

6. Decompress: Converts compressed columns into uncompressed representations.

7. Mask: Only values with the corresponding bit of 1 are exported.

8. Concat: Combine one or more projections arranged in the same order into a single projection.

9. Permute: Specify projection permutations in the order defined by the combination index.

10. Bit string operator: Checks for values of bit and bit variable types. (Ex: AND: &, OR: | , NOT: ~, etc.)

Query processing:

Queries in column stores can often select only a few columns from each table, reducing the total I/O of physical media. For column stores to respond to queries, projections must be combined using storage keys and join indexes. Within a segment, all data values in all columns must be associated with unique keys, and values in different columns with matching keys must belong to the same logical row.