1. Parallel Data Models
2. Assume a program P running in 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 the it can be easily divided into smaller tasks executing concurrently across multiple processors. What are the speed-up respectively? What are the optimal speed-up given infinitely amount of machines?

* **Speed-up with Limited Parallelism**: In a program P running on a single-processor system, if 20% of it can only be executed sequentially, and the remaining 80% is embarrassingly parallel and can be divided into smaller tasks executed concurrently, the speed-up can be calculated as follows: Speed-up (S) = 1 / [(1 – 0.8) + 0.8 / P] Where P is the number of processors used for parallel execution. If you plug in P=1 (single-processor system), you get S=1, which is the baseline performance. If you use more processors, the speed-up will increase, but it won’t be linear because 20% of the program is still sequential.
* **Optimal Speed-up with infinite Machines:** The optimal speed-up given infinite number of machines (P → ∞) can calculated using Amdahl’s Law, which takes into account the fraction of the program that cannot be parallelized. Speed-up (S) = 1 / [(1 - 0.2) + 0.2 / ∞] = 1 / (0.8) = 1.25. So in the limit of an infinite number of machines, the speed-up will be 1.25 times the original execution times

1. Describe and compare the pros and cons of the three architecture for parallel systems.

Shared Memory Architecture:

P: Easier to program and debug as all processors can access the same memory.

C: Limited scalability due to contention for shared memory, which can lead to bottlenecks

Message Passing Architecture:

P: Scales well with a large number of processors since communication can be more controlled

C: Programming can be more challenging due to explicit message passing

Data Parallel Architecture:

P: Well-suited for suited for regular and structured data-pa

C: Limited applicability to tasks that don’t exhibit data parallelism.

1. [ACID vs BASE] This set of questions are related to data consistency
   1. 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.

* What is CAP Theory?
  + The CAP theorem, also known as Brewers theorem, is a fundamental concept in distributed systems that deals with the trade-offs between three critical properties in a distributed database system: Consistency (C), Availability (A), and Partition Tolerance (P). It states that in a distributed system, you can’t guarantee all three simultaneously
* Explain CAP theory for the example cluster.
  + Consistency: This property ensures that all nodes in the system see the same data at the same time. With network partitions or node failures, maintaining consistency can be challenging. To achieve strong consistency, any write operation should not be considered successful until all nodes have been updated. When further research I learned, this can be difficult to achieve in a geographically distributed system, even more when there are network partitions.
  + Availability (A): Availability ensures that every request to the system receives a response, without guaranteeing the content of the response. In a distributed system, ensuring high availability might require accepting eventual consistency. This means that some nodes might temporarily have outdated data due to partitions or failures.
  + Partition Tolerance (P): Partition Tolerance means that the system continues to operate even when network partitions occur, causing communication failures between nodes. Achieving both consistency and availability during partitions can be challenging, so some distributed systems prioritize partition tolerance.
  1. 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”
     1. Add $200 to account B1:

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

* + - Atomicity: This operation is atomic as long as the database system supports atomic transactions. It ensures that the entire operation either succeeds or fails.
    - Consistency: The operation does not violate consistency as long as B1 exists, and there are no constraints or triggers that would make the account balance inconsistent.
    - Isolation: The isolation property ensures that this operation won’t interfere with concurrent operations on the same data. It may violate isolation if there are concurrent transactions attempting to modify account B1.
    - Durability: If this operation is committed, it should be durable.
    1. 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

* + - Atomicity: This operation is atomic as long as the database system supports atomic transactions.
    - Consistency: Consistency may be violated if A1 doesn’t exist or if the subtraction causes the account balance to become negative, violating integrity constraints.
    - Isolation: Similar to the first operation, Isolation might be violated if concurrent transactions modify account A1.
    - Durability: If committed, it should be durable
  1. (10) What is “BASE”? give an example of “BASE” data consistency model and compare it to ACID.
  + BASE is an acronym that stands for, Basically Available, Soft state, Eventually consistent. It represents a data consistency model that is often used in distributed and NoSQL databases as an alternative to the ACID model. BASE acknowledges that strict ACID guarantees might be impractical or overly restrictive in distributed systems where high availability and partition tolerance are critical.
  + BASE vs ACID
    - While ACID provides strong guarantees of data consistency, it can be rigid and less suitable for highly distributed and fault-tolerant systems. BASE, on the other hand, relaxes these guarantees to prioritize availability and partition tolerance, allowing for more flexible and efficient operation in distributed environments, but at the cost of potentially temporarily inconsistent data. The choice between ACID and BASE depends on the specific requirements and trade-offs of the applications or system being designed.

1. [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 a distributed systems technique that helps ensure data consistency in read and write operations, especially in the context of distributed databases or distributed systems. It works by using a voting mechanism to determine when a majority of nodes in the system agree on the state of data, which can then be considered as the truth.
* Breakdown of Quorum Consensus:
  + Data Replication: in a distributed system, data is often replicated across multiple nodes to ensure fault tolerance and availability.
  + Quorums: Quorum Consensus relies on the concept of quorums. A quorum is a subset of nodes that, when combined, is considered authoritative. In other words, it’s a group of nodes that can make decisions on behalf of the entire system.
  + Read Operations: When a read operations is preformed, the system contracts multiple nodes to retrieve the data. Each of these nodes can provide a different value if there is any inconsistency. However, the client can use the quorum concept to determine which value is considered the most up-to-date.
  + Write Operations: When a write operation is preformed, the system must ensure that the data is updated consistently across multiple nodes.
* Why Quorum Consensus works:
  + Majority Agreement: Key principle behind Quorum Consensus is that a majority of nodes must agree to ensure data consistency. This is crucial because it prevents split-brain scenarios where data could be updated independently on different nodes.
  + Fault Tolerance: By requiring a majority vote, Quorum Consensus allows the system to continue functioning even in the presence of network partitions or node failures.
  + Balancing Consistency and Availability: Quorum Consensus provides a balance between data consistency and system availability. It ensures that data is consistent enough to avoid conflicts while still allowing the system to operate even when some nodes are temporarily unavailable.
  + Configurable: Quorum sizes can be configured based on the desired level of consistency and availability. Systems can choose to have larger or smaller quorums depending on their requirements.

1. [Relational DB – Query Processing]

This question tests the understanding of basic relational database search operators. Consider a join. ⋈ 𝑅.𝐴=𝑆.𝐵. We ignore the cost of output the result, and measure the cost with the number of I/Os. Given the information about relations to be joined below:

Relation S contains 20,000 tuples and has 10 tuples per block. Relation R contains 100,000 tuples and 10 tuples per block. Attribute B is the primary key of S. In total, 52 blocks are available in memory. Assume neither relation has any index.

* Describe a block nested join algorithm. Give the cost of joining R and S with a block nested loops join.
* **Block Nested Loop Join Algorithm**

1. Initialize an empty result relation to store the joined tuples
2. Divide the available memory into two parts: one for holding blocks of relation R (outer relation), and other for holding a single block of relation S (inner relation).
3. Read a block of relation S into the inner relation buffer (memory)
4. For each block of relation R, the following must happen:
   1. Read the block of R into the outer relation buffer
   2. Perform a nested loop join by comparing each tuple in the block of R with each tuple in the inner relation buffer.
   3. If a match is found, add the joined tuple to the result relation
5. Repeat step 4 for all blocks of R and then continue to the next block of S until all blocks of S have been processed
6. The result relation now contains the joined tuples

Cost of Block Nested Join:

* Reading the blocks of R requires (100,000 / 10) = 10,000 I/O operations because each block can hold 10 tuples.
* Reading the blocks of S requires (20,000/10) = 2,000 I/O operations
* For each block of R, we need to compare it with the entire block of S in memory, which results in (10,000 \* 2,000) = 20,000,000 comparisons
* Since we have 52 blocks available in memory and we’ll use a portion of them for reading and holding blocks, we’ll need to perform multiple iterations of the join, which requires additional I/O operations to bring the necessary blocks into memory.
* The block nested loop join can be extremely inefficient for large datasets and is not suitable for use with large relations, as it has a high I/O cost . Other join algorithms like sort-merge join or hash join are more efficient.