CptS 415 | Assignment-04

Charles Nguyen, #011606177

1. Parallel Data Model

a. Amdahl's Law.

$$Speedup = \frac{1}{f + \frac{1-f}{s}}$$

where, $f = 0.20$ is fraction of the sequential part and s is the amount of parallel resources.

Time costs:

- 2 cores: $T_2 = 0.6T$
- 4 cores: $T_4 = 0.4T$
- 8 cores: $T_8 = 0.3T$

Speedups:

- 2 cores: $S = 1.\overline{6}$ times
- 4 cores: $S = 2.5$ times
- 8 cores: $S = 3.\overline{3}$ times

Upperbound of speedup for infinite number of cores:

$$lim_{x \to \infty} \frac{1}{0.2 + \frac{0.8}{x}} = \frac{lim_{x \to \infty} 1}{lim_{x \to \infty} 0.2} = \frac{1}{0.2} = 5$$

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

- Shared Memory:
    - efficient communication, because memory is accessible to all processors.
    - **low scalability** since the shared memory and network capacity end up becoming the bottleneck of communication.
    - even if memory cache is added for each processor, there is still the problem of data consistency whenever data needs to be updated across all processors.
    - upperbound of scalability is 32/64 processors.

- Shared Disk
    - high fault tolerance, because other processors can take over when one processor fails.
    - **better scalability** because data is now resident on disk without being dependent on memory.
    - is still reliant on network capacity.
    - upperbound of scalability is a couple hundred processors.
- Shared Nothing
    - **only queries** are passed through the network
    - theoretically **no upper limit to scalability**

2. [ACID vs BASE]: Data Consistency

a. CAP Theorem

> A system can have at most 2 out of 3 properties:
>
> - **Consistency**: each client can always read and write
> - **Availability**: all clients always have the same view of data.
> - **Partition Tolerance**: the system produces deterministic outcomes despite physical network partitions.

```
Assumptions:
  - a system with three geographically distributed servers S1 and S2 and S3.
  - data is partitioned across all three servers.
```

- **A**vailability and **P**artition Tolerence (forfeiting **C**)
    - data is always available for each server, but is not guaranteed to be up-to-date without *replication and verification*.
- **C**onsistency and **P**artition Tolerence (forfeiting **A**)
    - data needs to be consistent across all three servers, for which the system must perform *replication and verification*, during which the data is inaccessible.
- **C**onsistency and **A**vailability (forfeiting **P**)
    - data is always available and consistent for one server, but this state is not guaranteed to hold across all three servers, and thus needs to be *replicated and verified* for the other two.

b. Show ACID can be violated using the relation Accounts.

> A database requires 4 properties:
>
> - **Atomicity**: when an update happens, it is *all updated or nothing is updated*.
> - **Consistency**: the states of various tables must be consistent (relations, constraints) at all times.
> - **Isolation**: concurrent execution of transactions produces the same outcome as if done sequentially.
> - **Durability**: once commmitted, the outcome of a transaction is immutable to problems like power outage, etc.

The commit has two statements, so they are not atomic. This commit is not resistent to a power outage, which might cause only one statement to be committed successfully, regardless whether they were submitted sequentially or concurrently. This situation in turn cause the accounts to be inconsistent.

c. ACID vs BASE

BASE database model fundamentally forfeits **C**onsistency and **I**solation for **A**vailability:

> **B**asically **A**vailable,
> **S**oft state,
> **E**ventually consistent

A BASE model allows for:

- stale data (weak consistency)
- prioritizing availability
- best effort
- approximate answers
- simpler and faster than ACID

3. Quorum Consensus

Quorum Concensus is a voting model to improve fault tolerance and consistency. A quorum is the minimum number of majority votes to win any transaction commit. A quorum must be defined for every system, typically $W + R > N$ where N is the number of servers in the system.

This model contains two sequential operations, **put** and **get**, and a final vote count. A **put** request is sent to multiple servers so that in the case one server fails other servers still hold copies of data. Next, a **get** request is sent to multiple servers containing the redundant data. Finally, the commit is granted if and only if the sum of successful **put** and **get** requests is larger than the total number of servers.

This model works because it satisfies all ACID properties.

4. Column Store

Column Store (CS) offers significant speedup over *Row Store* (RS) for some applications. A column is atomic and pure by virtue of the column label (property). In memory, all rows in a column stored sequentially without being mixed up with other properties like in RS. This means that a columnar dataset pertains purely to that column and do not contain data from other columns (irrelevant data). This also means the fetched data footprint is smaller when loaded.

CS offers support for the following operators:

- Select: same as relational algebra; produces a `Bitstring`
- Project: same as RA
- Join: joins projections with respect to predicates
- Aggregation: SQL-like aggregations
- Sort: sorts all columns of a projection
- Decompress: expands columns to uncompressed representation
- Mask`(Bitstring B, Projection Cs) =>` emits only values whose coressponding bits are 1
- Concat: Combines one or more projections sequentially without changing order into a single projection
- Permute: Permutes a projection according to the ordering defined by a join index
- Bitstring operators: BAND -- bitwise AND, BOR -- bitwise OR, BNOT -- bitwise complement