Solutions for Assignment 1

Rodrigues, Diogo
MTK 03770446
diogo.rodrigues@tum.de

TUM – Cloud-Based Data Processing 2022/23
3rd November 2022

My answers for Assignment 1.

Section 1: Snowflake - Architecture and design decisions

Snowflake - System Design

(i) For what workloads is Snowflake built for/best-suited?
A: Snowlake is best-suited for analytical workloads, specifically for cloud-native data warehousing of large amounts of data.

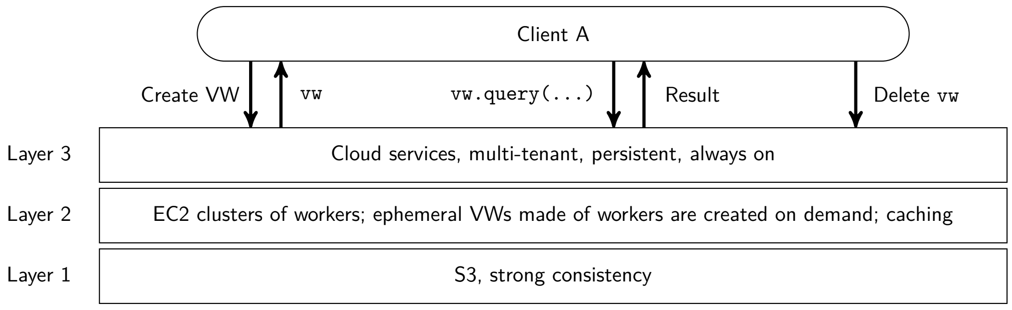
(ii) What are the functional and non-functional requirements of Snowflake?
A: The functional requirements of Snowflake are that:

- It provides a pure Software-as-a-Service experience
- It provides the persistent data storing functionality of a data warehouse
- It has support for ANSI SQL and ACID transactions, working as a relational database
- It has support for semi-structured, schema-less data
- It has an ODBC interface
- It has a graphical interface that allows users to manipulate and query their data

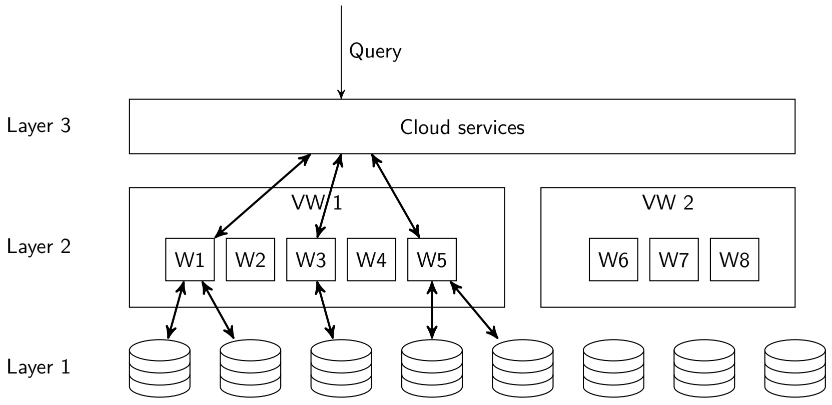
The non-functional requirements are:

- To be able to horizontally scale storage and compute independently, without impact to availability or performance.
- High availability in face of large-scale failures or system upgrades.
- Durable to node failures so as to avoid data loss or unavailability
- The system should be multi-tenant and only use as much resources as strictly required, so it is cost-efficient.
- End-to-end encryption to guarantee security.
- It should be able to handle queries that generate large amounts of temp data, even if the amount of temp data is far larger than available primary memory (RAM) or available disk space of a single node.
- It should be as fast as possible at processing operations involving large amounts of data (e.g., large intake of data, queries that involve large amounts of data).

(iii) Draw a high-level system diagram of Snowflake
A:


Client A
Create VW
vw
vw.query(...)
Result
Delete vw
Layer 3
Cloud services, multi-tenant, persistent, always on
Layer 2
EC2 clusters of workers; ephemeral VWs made of workers are created on demand; caching
Layer 1
S3, strong consistency

(iv) Draw a second diagram visualising how a submitted query is handled by the system.
A:



(v) How are performance limitations of S3 tackled by Snowflake?
A: the main performance limitation of S3 is the large access latency associated with each I/O operation, which is derived from the fact S3 storage is made using secondary memory. This performance limitation is tackled by Snowflake by having each worker node maintain an extensive cache of table data. Performance is further improved by having each worker node be responsible for certain input files, so redundant caching across different worker nodes is avoided.

(vi) Cost provisioning is important when working in the Cloud.

1. What are the main cost elements included in Snowflake? Which is the most expensive? (hint: https://medium.com/wise-engineering/3-main-elements-in-your-snowflake-bill-45331ab7b224)
   A: The main cost elements in Snowflake are storage and compute resources. Compute is especially expensive and accounts for around 90% of the costs.
2. What are the costs included in the storage layer of Snowflake? (hint: https://aws.amazon.com/de/s3/pricing/?nc=sn&loc=4)
   A: The costs in the storage layer of Snowflake include:
   - The cost of storage per GB.
   - The cost of each request type.
   - The cost of bandwidth from and to Amazon S3.
   - The cost of using management and analytics tools of Amazon S3.
   - The cost of replication, which is the extra cost of storage per GB plus additional charges for using inter-region bandwidth and for the guarantee that the storage layer will handle replication automatically.

Section 2: Dynamo - Architecture and design decisions

Amazon Dynamo - System Design

(i) For what workloads is Dynamo built for/best-suited?
A: Dynamo is best-suited for transactional workloads, where queries are simple but arrive at a very high rate, and must be answered as fast as possible.

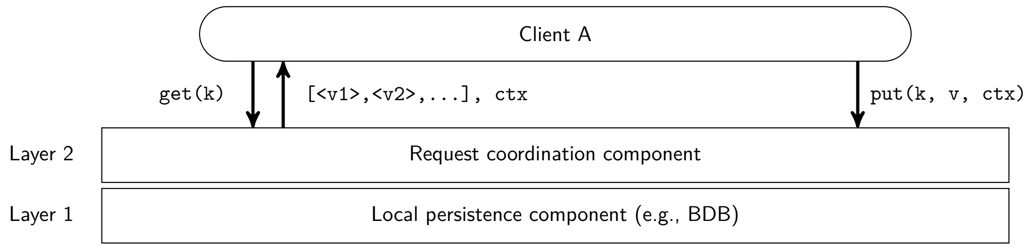
(ii) What are the functional and non-functional requirements of Dynamo?
A: The functional requirements of Dynamo are that:

- It provides a key-value store service for arbitrary data.

The non-functional requirements of Dynamo are that:

- It is highly available, even if at the cost of consistency.
- It uses resources in an efficient way, so it can scale according to demand.
- It handles large numbers of requests simultaneously (hundreds of requests per second, hundreds of thousands of sessions open at any given point).
- It can run in an heterogeneous network, where machines/nodes have different computing resources.
- The system is symmetric, so there is no node hierarchy nor is there a moment where a specific node has more responsibilities than others.

(iii) Draw a high-level system diagram of Dynamo
A:


Client A
get(k)
[<v1>,<v2>,...], ctx
put(k, v, ctx)
Layer 2
Request coordination component
Layer 1
Local persistence component (e.g., BDB)

Section 3: Distributed Systems Questions

CAP Theorem

(i) Where do the following stand in terms of the CAP theorem, and why?

(a) Snowflake | **A:** Snowflake stands in the same place as S3 does, given that Snowflake cannot provide more guarantees than the lowest layer. Because S3 guarantees AP with eventual consistency, Snowflake also stands in AP.

(b) Amazon Dynamo | **A:** AP, because Dynamo sacrifices consistency in the event of a network partition by allowing concurrent writes and requiring conflict resolution at the application level.

(c) Relational database management systems (e.g. PostgreSQL) | **A:** CA, because RDBMS's enforce very strict consistency and availability, at the cost of not being suitable for distributed systems (i.e., RDBMS's cannot handle network partition well).

(ii) Describe two scenarios in which Amazon Dynamo gives inconsistent answers. The one scenario should involve a node fault and the other should include no node faults.
A:

- First scenario: consider a ring with nodes A, B, C, in the ring by this order. Consider $N = 3$, $W = 2$, $R = 2$. Key k is coordinated by A. In the initial state, A, B and C contain the mapping $k \to x$. The value of key k is modified to y, so A and B now contain the mapping $k \to y$ (because $W = 2$, we do not need to update the mapping in C). A network partition occurs, and the nodes are partitioned into $\{A, B\}$ and $\{C\}$. A request arrives at C asking for the mapping of k. Although $R = 2$ and there are not two nodes that are part of the quorum for key k (because the only members of the quorum of k that C can reach is itself), C privileges availability and returns the best answer it can: $k \to x$, although the mapping was already changed to y.
- Second scenario: consider the same initial state as the previous scenario. The value of key k is modified to y. However, according to the article, the `put` request may return before the replicas are updated in other nodes. This means there is a possibility that the `put` request returns before replicas at B or C are updated. If a `get` event arrives in this short span between the time when `put` returns and the replicas are updated, there is a read quorum of $\{B, C\}$ claiming that $k \to x$, although it was already changed to y.

Data Center Failure - Disaster Recovery

(iii) How does Snowflake preserve consistency in case of a data center failure? How does Dynamo preserve availability?
A: We are assuming that S3 uses replication. Snowflake does not preserve strong consistency, because the lowest layer (S3) only guarantees eventual consistency. Therefore, in the event of data center failure, Snowflake can only guarantee strong consistency with unbounded latency if it retries until the required nodes are back online; or establish a timeout, in which case it guarantees strong consistency with bounded latency but sacrificing availability by allowing an operation to fail after N retries, or after a time span Δt.
A: Dynamo preserves availability by always allowing reads even if the quorum requirements are not met (sloppy quorum). Dynamo furthermore always allows writes through hinted handoff, where a node B that is not part of the write quorum is required to temporarily store writes until the node A it is replacing comes back online, at which point B transmits all the hinted writes to A.

Design flexibility

(iv) Amazon Dynamo offers eventual consistency guarantees. What would you change in its design in order to make it a strongly consistent system? What would be the tradeoff?
A: The easiest way would be to change Dynamo's sloppy quorum into a regular quorum, where we place stricter definitions on what nodes are part of the quorum or not. This would provide strong consistency due to the guarantees quorums provide, but would require sacrificing availability or potentially having availability with unbounded latency.