Cloud-Based Data Processing
Assignment 1 - High Level System Design

Handout: **9^{th} November 2020**
Due: **17^{th} November 2020 by 6pm**
Discussions: **11^{th} and 18^{th} November 2020 in the tutorial**

1 Introduction

This exercise aims to teach you how to analyse and describe a system at a high, abstract level while being able to go into depth on specific topics if asked to. It prepares you for the next exercise, where you are asked to design a system given a use-case.

The exercise is organized around two papers [1, 2] which you find on Moodle. One introduces the analytical, cloud-native data warehouse *Snowflake* and the other describes Amazon's Dynamo key-value store. Both papers are highly influential for later cloud systems. We strongly recommend reading the background on distributed systems from the tutorial slides before reading the Amazon Dynamo paper.

In section 3 and 4 you are asked to apply a typical recipe of high-level system analysis to both systems. After we ask two questions comparing the two systems, this should help you to tease out the alternative design choices taken to solve the different requirements of both systems. In the last two sections, we put the focus on two aspects which strongly influence every cloud-based system design: distributed systems challenges and cost analysis.

Some exercises are marked *Optional*. We will discuss them last in the tutorial but are there to give you additional room to think and prepare you for the type of questions you may get on the exam.

2 Submission guidelines

Questions should be answered briefly in several sentences or bullet-points. As often with high-level systems discussion, there are many answers possible and we find that discussion can be more important than the concrete answer. Please be ready to present your answer to the class and be able to outline the advantages and disadvantages of your choices.

The first tutorial is meant to pose questions and discuss possible answers, so please go through the exercises before.

Fill out the answer to the questions directly in the PDF or by hand on a printed version. Drawings should be done on one A4 page and appended to the end of the hand-in. You can use this webpage `https://combinepdf.com/en/` to merge multiple PDF files.

Send an email to mailto:per.fuchs@cs.tum.edu with a **single** PDF attachment. If you fill in the PDF directly, also send all answers in a *.txt* file - some PFD reader combinations are known to have issues displaying form-fields correctly. Diagrams should be provided as PDF only. In total, I want **one** PDF and, if forms are filled in electronically, **one** additional text-file.

3 System Design: Snowflake

Often it helps to discuss a system design following a common recipe by (1) describing the workload or use-case, (2) deducing the functional and non-functional requirements and (3) designing a system that fulfils all requirements. One useful way to visualize the system's design is by drawing a diagram covering their most important components and interaction between them. The process is described on slide 37 of the first lecture.

Focus only on what is written in the paper. There's no need for further research on use-cases or requirements not named in the paper. However, if you are not familiar with the workloads targeted or some of the terms used in the papers, a background research is necessary - feel free to ask in class.

3.1 Describe the use-case and workload that Snowflake was built for.

3.2 List the functional and non-functional requirements of Snowflake.

3.3 Draw a high-level diagram of Snowflake's system design.

Note: do not copy the diagram from the paper. Include the interactions between the three layers of the design, e.g. by showing their interface.

4 System Design: Amazon Dynamo

Now, we do the same for Amazon Dynamo as in section 3 for Snowflake.

4.1 Describe the use-case and workload that Dynamo was built for.

4.2 List the functional and non-functional requirements of Dynamo.

4.3 Draw a high-level diagram of Dynamo's system design.

Note: do not copy the diagram from the paper. Focus on visualizing the difference between the logical and the physical hash ring. Show the N-nodes as well as the *preference-list*.

5 Comparing the two systems: Snowflake and Dynamo

5.1 Where do both systems stand with regards to the CAP Theorem [3]?

5.2 Design a key-value store with decoupled storage and compute.

Imagine a system which uses Amazon EC instances to answer requests of the type *get_value_for(key)*. One could build it by using x Amazon EC instances and x files stored in Amazon Elastic File system. The keys are partitioned into x buckets by their hash value. Each file stores a B-Tree of its key-value pairs. Each EC instance opens one file and serves its keys to clients.

Background on EFS: `https://docs.aws.amazon.com/efs/latest/ug/whatisefs.html`.

1. Visualize the described system with a diagram.

2. Discuss the advantages and disadvantages of this design over an in-memory key-value store like Dynamo? In particular, consider how durability, consistency and availability requirements have been pushed into the storage layer. Describe how faults could be handled in the new system.

3. Give a simple design on how to make the key-value store explained above elastic.

4. (Optional) The example system uses Amazon EFS instead of Amazon S3. Explain this design choice by arguing that a key-value store has different requirements for storage than analytical workloads as handled in Snowflake, which uses S3. Mainly consider the costs and that in S3 you pay on a per access basis.

6 Focus: Distributed Systems Challenges

6.1 Describe scenarios in which Amazon Dynamo gives inconsistent answers.

Hint: go step by step by giving a comma-separated list of events that lead to an inconsistent answer. E.g. write value x to key k_1 on node w, node w crashes before xyz, read k_1 on node u returning value not_x, inconsistency occurred. The trick here is to give a minimal example of an execution trace containing only the relevant state of the system. Generally, this method helps in understanding fault-tolerance.

1. Scenario with one node fault:

2. Scenario without any node fault:

6.2 How to make Dynamo consistent?

There is one simple change in Dynamo's design which turns it into a CP system [3]. In other words, show how to make it consistent by giving up on some availability.

1. What is the change in the design?

2. It is now possible to trade-off availability and latency. How?

6.3 (Optional) Visualize each challenge named in table 1 of the Dynamo paper in your systems diagram.

6.4 (Optional) Where are the distributed system challenges hidden in the Snowflake paper?

The paper of Snowflake does not cover many distributed systems challenges. However, that does not mean they magically disappeared being afraid of the excellent engineers at the company. Describe for each of the layers below which consistency and availability guarantees it pushes to the other layers. Do not focus on how this is implemented, rather on how the interaction between layers composes a complete, consistent, and highly-available system. Hint: refer back to your system diagram.

1. Consistency and availability of the storage layer (Amazon S3). Background: `https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html`

2. Consistency and availability of the Cloud Services layer:

7 (Optional) Focus: Costs

7.1 (Optional) Analyze the costs of running a Snowflake's service.

The goal of this exercise is to get a better understanding of the costs of running Snowflake and identifying which costs are paid directly by the customer or indirectly by the pricing of the service.

Which costs arise in the storage layer? Background: `https://aws.amazon.com/de/s3/pricing/?nc=sn&loc=4`

Who pays the costs arising at the storage layer?

The customer does not pay anything for the Cloud Services layer as long as the use is in *normal* proportion to his compute layer costs. So, most customers do not pay for the Cloud Services layer. Why is that a good deal both for the customers and for Snowflake? Hint: think of system utilization.

7.2 (Optional) Why does separating compute and storage pay off financially?

Imagine an alternative where the compute nodes also hold the data. Hint: double-check the pricing and persistence of hard disks on the AWS EC machines.

References

[1] Dageville et al. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, pages 215–226, 2016.

[2] DeCandia et al. Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220, 2007.

[3] Gilbert et al. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM Sigact News*, 2002.