Cloud-Based Data Processing
Assignment 3 - A physical design for an URL shortener service

Handout: **24th November 2020**
Due: **01st December 2020 by 6pm**
Discussions: **25th November and 02nd December 2020 in the tutorial**

1 Introduction

In this exercise, we plan to implement the design from the last exercise with the service offerings of Amazon, e.g. one of its transactional databases for the `key-db` or one of its three key value stores for the caching and main database. The aim of this exercise is to (1) decide which service to use for which component and (2) to decide how many instances we need of each Amazon service for our component.

This exercise is difficult for two reasons. First, students sometimes feel they miss the necessary background to estimate how many machines are needed to handle a workload. All necessary background is given on this exercise sheet in the Section 2. If you find its hard to give an exact number of machines necessary for a specific component, we recommend to think in terms of possible bottlenecks, e.g. memory or computational power. You should enumerate them and other likely bottleneck and discuss which is likely to be the limiting factor for the workload in question. If this is not enough to come to a concrete number of machines, explain which questions need to be answered and how you would go about this.

Second, all the questions are very open. This is unusual for academic exercises but very typical for system design. It is an inherent challenge in system design and we are aware that it is hard. However, there is a structured way of tackling such questions:

1. Define the system or component you are talking about by its functional and non-functional requirements.
2. Clarify which estimates you need to design the system.
3. Start with a simple, intuitive first design - the first that comes to your mind.
4. Enumerate up to 2 alternatives addressing problems with the first solution.
5. Explain the advantages and disadvantages of each compared to each other.
6. Build an argument which convinces the reader why you recommend a specific alternative.

2 Background

We assume a good understanding of the model solution to the last exercise as described on Grokking the System Design Interview and shown in Figure 1. The model solution is the system design to implement, you should think *inside* of it and *not consider alternatives*. We diverge from the model solution only in the design of the `key-db` and explore multiple alternatives.

Given the use-case, it would make sense to build a multi data-data center system to be geographically close to our customers and to provide higher availability. **However, for the sake of the exercise, we assume that we serve all customers from a single data-data center and do not require the physical design to have machines in different regions.**

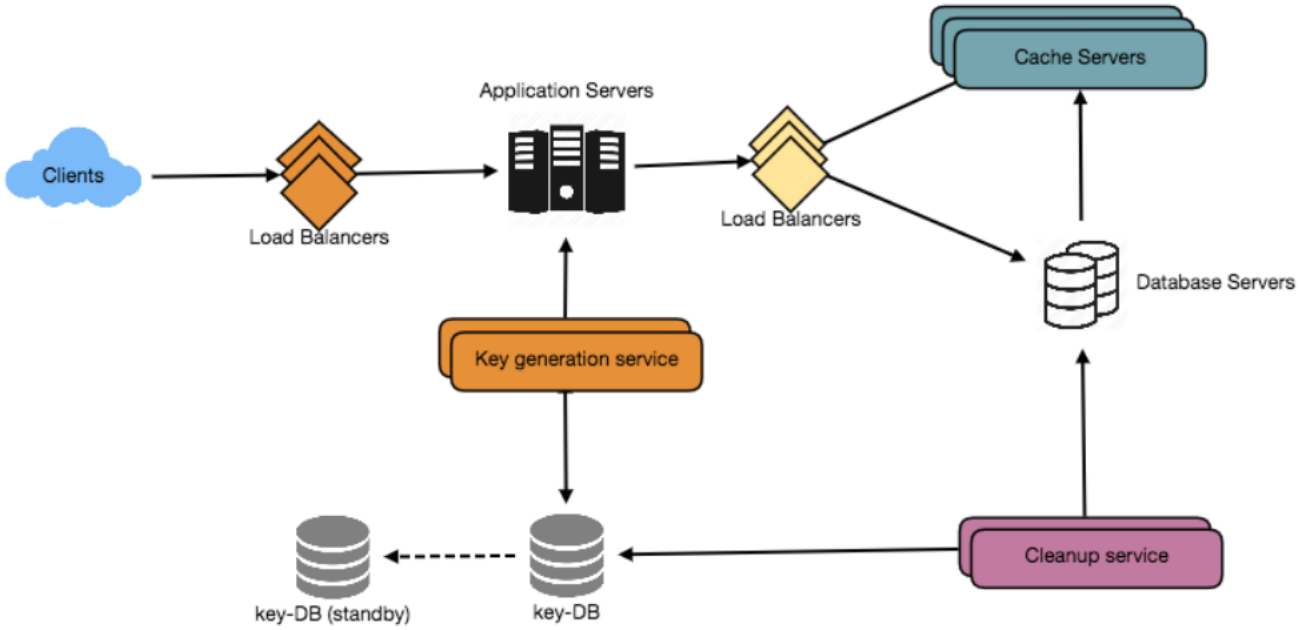


Figure 1: Detailed component design for URL shortening

Part of the exercise is to gain an understanding of Amazon service offerings and their pricing. We expect you to do background reading on the functionality and pricing scheme of the services listed below. Each page also includes a link for pricing.

- Amazon DynamoDB
- ElastiCache
- RDS: Relational Database Service
- EC2
- Elastic Load Balancing

You can assume that a transactional database can handle much more than 200 queries per second and a key-value store more than 20k queries per second.

We defined availability as the most important non-functional requirement for our URL shortener. Therefore, we focus on fault tolerance in this exercise. There is some background on it below:

- Fault-tolerance can be defined as the number of nodes that can fail until the whole component becomes unavailable. One describes this idea by stating that there are N nodes and that the component can tolerate 1, 2 or M failures. Such a component is called $N - M$ fault-tolerant.
- In particular, $2-1$ systems are relevant in practice if the only reason to have more than one machine is fault-tolerance.
- One way to reason about the degree of fault-tolerance needed is to discuss *mean-time-between-failure* ($MTBF$) vs *mean-time-to-repair* ($MTTR$). One can argue that a system with a high $MTBF$ but low $MTTR$ can have a lower M than one where the relationship between these metrics is reversed.
- Types of storage systems react differently to faults: ephemeral storage is lost, persistent storage survives node-failures but not necessarily disk-failures, and some persistent storage can survive single or multiple disk-failures, e.g. RAID systems.

2.1 Main key-value store

What guarantees are needed for the component labeled *Database Server*? Recall that this component is a key-value store and assume that we want to use Dynamo as described in the paper from the first exercise. Do not consider alternatives for this part of the exercise. Do not change Dynamo's design but describe difficulties which need to be addressed by other components in our design when using Dynamo.

1. Dynamo is an AP system. Which kind of inconsistencies can arise for our workload and how can we circumvent them? Hint: our workload is *append-only*.

2. Which configuration of N, R and W would you choose and why? Consider latency and availability arguments.

2.2 key-DB

In this part of the exercise, we design the component storing unused keys and the key deletion index. This is a transactional database. In the model solution, it is replicated via a synchronous, master-slave setup. Please, consider master-slave replication as well as at least one alternative design.

1. What would happen if the key-db is unavailable? Is this acceptable? If not, **how many** machines do we need for redundancy?

2. Would you use replication, partitioning or both? Explain your answer.

3. Which durability guarantees do we need for the storage of this component? Consider ephemeral storage, persistent and fault-tolerant persistent storage.

4. Describe the design you would recommend for the `key-db` component. This can be the same as in the model solution but argue why you think it is better than a second alternative described by you.

2.3 Cache Servers

What do we require from the caching layer? Consider fault tolerance, partitioning and cost. Then describe which hardware resource is the bottleneck for a caching server and **estimate the number of machines needed**. For this estimate, consider the hardware resource bottleneck as well as necessary fault-tolerance. Hint: the pricing scheme of Amazon AWS for caching machines is linear for a specific hardware resource.

2.4 Application Servers

Discuss the application servers component along the same lines as you did for the components before. In particular, estimate the hardware resource bottleneck and give an idea what you would need to estimate to answer how many `Application Servers` are needed.

2.5 Are there any components which should be supported by elastic (on-demand) instances? Does the design given allow it?

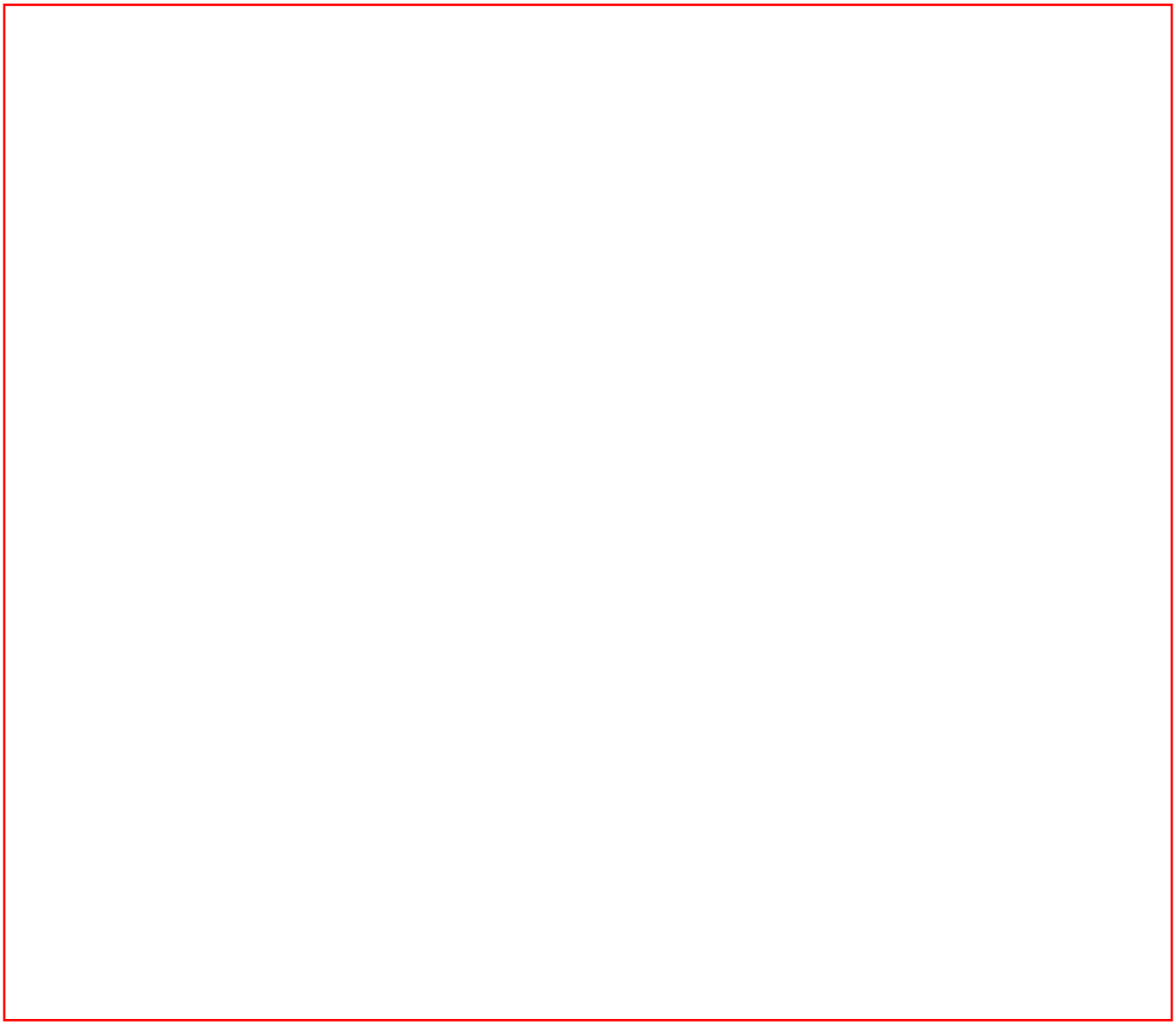
2.6 Map the services to physical resources

Please, note that Figure 1 shows both components and services. Components have certain hardware attached to them, e.g. Database Servers or Application Servers. Services are pieces of software and should be mapped to run on a specific server. Defining this mapping is part of this exercise. Collocating services makes sense to reduce latency and increase resource utilization. However, using different servers per service allows to scale each service separately and guarantees resource isolation.

Which services can be co-located and which should be located on separate machines?

2.7 Visualize the physical design

Draw all machines used or indicate if you want to use a variable number of machines for some components. In this case, give a ballpark estimate for how many machines each component should have.



3 Use Amazon AWS offerings to implement your design

We now implement the physical design you came up with by using the Amazon AWS offerings. Furthermore, we calculate the annual costs of running the service as a whole. It is recommended to complete this exercise in two parts. First, find the best-suited offer from Amazon to support each component. Amazon provides multiple options for some components (e.g., AWS offers three key-value stores: Redis, Memcached and DynamoDB). Second, calculate the costs of running the URL shortener each year. We provide an Excel sheet that you can use as a template. If you find that multiple offerings are a good choice for one component, add both alternatives to the pricing calculation.

Consider at least the Amazon offerings given in the beginning of the exercise. You can consider also others if you want to but it is not necessary.

Please use Amazon *US East (Ohio)* as the region for pricing.

AWS allows you to pay for most offers on-demand or reserve a contingent beforehand. Prices for reserved contingents are much lower than paying for the same component on demand which comes at the cost of not being elastic.

3.1 Which offer would you use for which component, and (shortly) why?

1. Database servers (kv-store):
2. Cache Servers:
3. key-DB:
4. Application Servers:
5. Load Balancer:

3.2 Calculate the yearly costs

Fill out this Excel Sheet.