DV1566 - Project Proposal

Group	17
Student 1	Isac Svensson
Student 2	Robin Stenius

1. Problem description

Computational stress on a system can cause significant performance loss, even with good algorithms and techniques, when the increase for said computation is under high demand by a lot of users. The average user does not care about the underlying system executing the computation, but rather only wants it done fast and on demand. This means the underlying system needs to be responsive to fluctuating demands at any given moment.

We want to implement a system which can handle ever increasing demands on sorting algorithm computations. This means the system should use some auto scaling ability with respect to the increasing demands on the system, meaning it must scale the system as a whole and redirect incoming computation demands seamlessly.

2. Proposed solution

Our proposal aims at satisfying the scalability and high availability of computation.
We would like to use the Amazon API Gateway together with AWS Lambda in order to tackle these challenges. The Amazon API Gateway and AWS Lambda can take one million requests for free per month with the AWS Free Tier subscription.

We will implement the bubble sorting algorithm in python, which will sort incoming arrays of various lengths and element sizes. This would be a plain and simple traditional sorting. After finished sorting it might send a response back with the sorted array or simply an acknowledgement of finished sorting.

The goal of the sorting algorithm is to sort any incoming array with integers into ascending order.

3. Test plan

We plan to use PyTest for testing. By sending requests to the API with multiple threads we will simulate multiple concurrent users.

We will focus on testing scaling, looking at how and when up/down-scaling is done, and availability when scaling. This can be done by observing AWS CloudWatch.

Checking that computations work as specified by returning the finished computation from the AWS Lambda back to the request source.

We will test the scalability over time via iterations by a combination of multiple threads to simulate multiple concurrent users, where each user has the same array size for each iteration but both number of concurrent users and array size will be increased over time.

We will use PyTest to create the threads necessary where each thread sets up the simulated user array, sends the request to the API and waits for a response before finishing. This while measuring time for different levels of concurrent users and checking that we got the correct result (which should not be a problem).