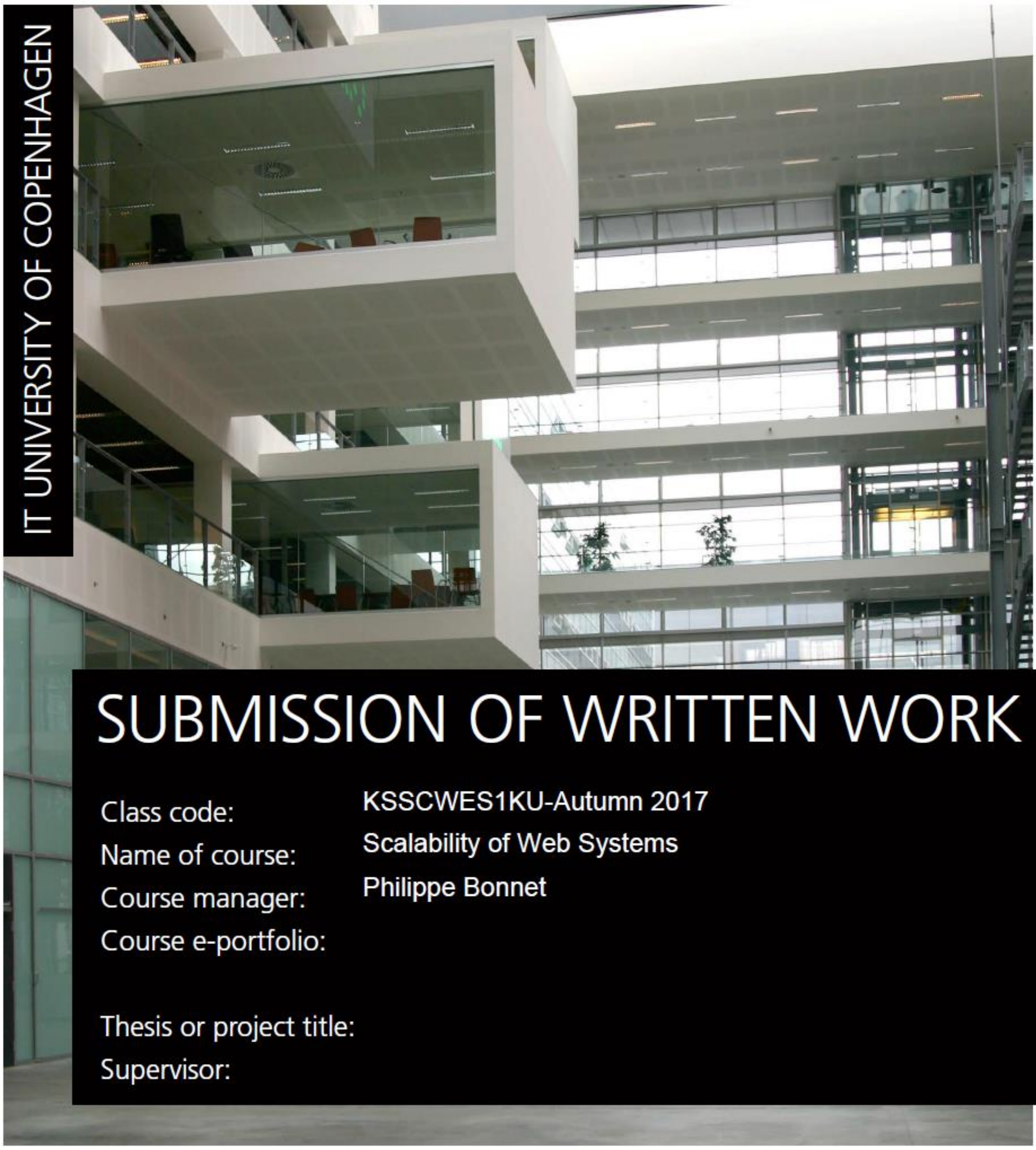
dttn@itu.dk
4/1/2018



SUBMISSION OF WRITTEN WORK

Class code:	KSSCWES1KU-Autumn 2017
Name of course:	Scalability of Web Systems
Course manager:	Philippe Bonnet
Course e-portfolio:	
Thesis or project title:	
Supervisor:	

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Dennis Thinh Tan Nguyen	01/04-1993	dttn@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Dennis Thinh Tan Nguyen
dttn@itu.dk
4/1/2018

Question 1

A

One of the main challenges was understanding the domain and how the satellite dataset was structured in Google Cloud Platform and how to query for the correct satellite data when a user was to input some latitude and longitude coordinates. Since the data are sorted and stored in their specific granule based on their MGRS tile, we were to find a way to convert latitude and longitude coordinates. For this, we called another third-party web service which provided an API that allowed us to get the MGRS value based on some given latitude and longitude coordinates.

Once the MGRS value was extracted another challenge was to query for the data. For this, we utilized Big Query which provided an interface that allowed use to use a SQL-based input to query the satellite data based on the extracted MGRS value. Yet, Big query only returned which granules the images were in based on the MGRS value and not the exact images. Thus, another challenge was to extract the images based on the returned granules from Big Query. Since the satellite data was hosted on Google Storage, we could utilize their API to fetch the requested images based on the list of granules returned from Big Query.

This solution performed well if one were to extract satellite data from a single point, however, if one were to extract satellite data based on a range such as a specific area on the earth, then a huge number of granules and images had to be extracted. Thus another challenge was how to scale the system, so it was able to cope with the large data output while still being well-performing. Here we utilized go routines and channels to concurrently extract the images from the granules.

B

The absence of Big Query would have a huge impact on the system since the system, and its implementation is highly dependent on Big Query as it returns the granules which we need to extract the satellite data from Google Storage. Consequently, we need to reimplement the system such that it satisfies the context and structure of the other given platform.

Question 2

A

Depending on the underlying architecture of the system, if the system utilized the traditional 3-tier architecture, one would use a load balancer that would distribute each request to the underlying applications whose purpose is to fetch the satellite data from Google Cloud Platform.

If the system used micro-services, then one would scale it across nodes, clusters or even data centers if the number of requests or the size of the data is massive. Assuming the system is implemented on a cluster one may scale the system horizontally by increasing the number of machines.

B

A RPC-based service may be equally scalable as a REST-based service as both are in simple terms HTTP interfaces to some application service. How well they these application scales depends solely on how they are implemented, thus a RPC-based may scale worse or better than a REST-service and vice versa depending on how they are implemented.

However, a REST-service may be preferred due its underlying constraints of its interfaces such as everything is considered resources and are identified through URI and also uniformed interface with stateless interactions[1]. It is the abstraction of the service that makes REST-based service different from RPC-based service and therefore may make REST-based service more scalable. By way of example, RPC-based services are composed of fine-grained user-defined operations.

Each application within the service may hay different interfaces with their semantics and operating parameters. Thus, any developer or system administrator must understand the semantics of each interface. This may not be a problem on smaller systems, but on large-scale systems with many applications, the number of different interfaces and semantics to understand may pose a challenge when one is to scale up the system.

This problem may not exist in a REST-based service as one of its constraints are that all applications use a uniformed interface. That is all resources that the application provides expose the same interface to the clients within the REST architecture.

Also due to the stateless interactions provided by REST, no states are stored within the server, and thus all requests must contain all required states to understand such requests. Thus stateless interactions may reduce the cost to enlarge the system. Whereas RPC-based services some interactions may be stateful and one may then implement a system that can maintain the states. This may increase the complexity of the system and thus reduce the scalability of it.

In conclusion, REST-based systems may scale better than RPC-based systems due to its constraints to the underlying interfaces, but that may not guarantee that REST always performs better than an RPC-based system as the scalability of both systems depends entirely on how they are implemented.

Question 3
A

To eliminate single points of failures and increase availability on can add redundancies in the system. Thus there are two ways to do so: Cloning and Partitioning.

When cloning, all data is replicated on all nodes within the system. Hence, one may avoid backup systems and consistency requirements since all data are replicated. To scale up one can just remove and add clones on demand and avoid any dead or corrupt clones. To avoid single point failures, one depends on having at least one clone. However the problem with this method is that it only works on stateless data therefore if any stateful data is required, then this method cannot be applied as one needs to maintain the consistency of the states of the data.

So, another way of eliminating single point of failures when one utilizes stateful data is to use partitioning. The stateful data is partitioned across partitions where each partition consists of a pair of nodes. One node is a backup and another node is the operating. Thus, if the operating node fails, the backup node can take over until the operating is fixed. If both nodes fails then the data is lost. The consistency requirement here is the data in the backup must be equal to the operating node. To scale up, one can simply add or remove partitions and then reparation the data.

Both methods as their pros and cons. For cloning, no backup or consistency requirements are needed but does not work with stateful data. For partitioning, stateful data can be used but requires backup and consistency requirements are needed which may increase the complexity of the system.

[1] REST: An Alternative to RPC for Web Services Architecture – Feng X., Shen J, Ying F. - 2009 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5339611

B

The role of replication is to increase availability of the data by increasing the number of the data across many nodes. With the master-slave replication approach, one has a pair of master systems, one for operation and one for backup, which acts as an interface for any requests to the system. The master has a set of nodes that contain the data. The master knows which node contains which data and fetches it from the given node. The problem with master-slave is the master may in itself be a single-point of failure if it fails since all requests are routed through it.

With Peer-to-peer replication all data is replicated across all nodes. Each node may all be connected and may also take any requests, whereas, in the master-slave, only the master may take requests. One does not know the exact location of the data since each node only knows its dataset. If the data does not exist on a given node, the request is propagated to another node, until the request reaches a node which contains the data. This approach is more robust than the master-slave approach but is also more complex due to its implementation.

Question 4

A

When a new request reaches the IndexHandler, a new goroutine is automatically created for such request to allow the service concurrently handle other incoming requests. The handler processes the request by generating the string “Welcome to the image service…” and calls the template engine which would then generate the necessary HTML code. Finally, the HTML code and the string is bundled in an HTTP-response and sent back to the client

Go enables scalability for multiple requests because it utilizes the HTTP/2 network protocol. HTTP/2 decreases latency to improve response time by compressing the HTTP headers thus reducing the overhead of a request. Also it allows multiplexing multiple requests over a single TCP connection to avoid creating a new TCP connection for each request and finally pipelining of request to avoid the browser await a response for each request before another one is sent. Thus improving performance when handling multiple requests compared to when it was using HTTP/1 which could only handle a request at a time**.**

Go utilizes goroutines which is a piece of work that can be executed concurrently with other goroutines in the same address space. That is the runtime does the scheduling and not the OS. All the goroutines are multiplexed and scheduled by the runtime onto threads which are allocated to the runtime by the OS.

Compared to Unix OS threads and processes, the OS schedules the threads to the processes which is similar to how the runtime schedules the threads to the goroutines.

B

The interface mechanism describes behavior and allows dynamic typing in go. Thus if one uses functions that take these interfaces one can introduce new code while still being able to use existing functions by implementing and satisfying methods required by such interface. Thus, any concrete implementation of handlers that satisfy the HTTP-handler interface be registered as handlers to HTTP.HandleFunc function.

Thus, interfaces enable sharing common code across web services since it allows dynamic typic in the implementation.

C

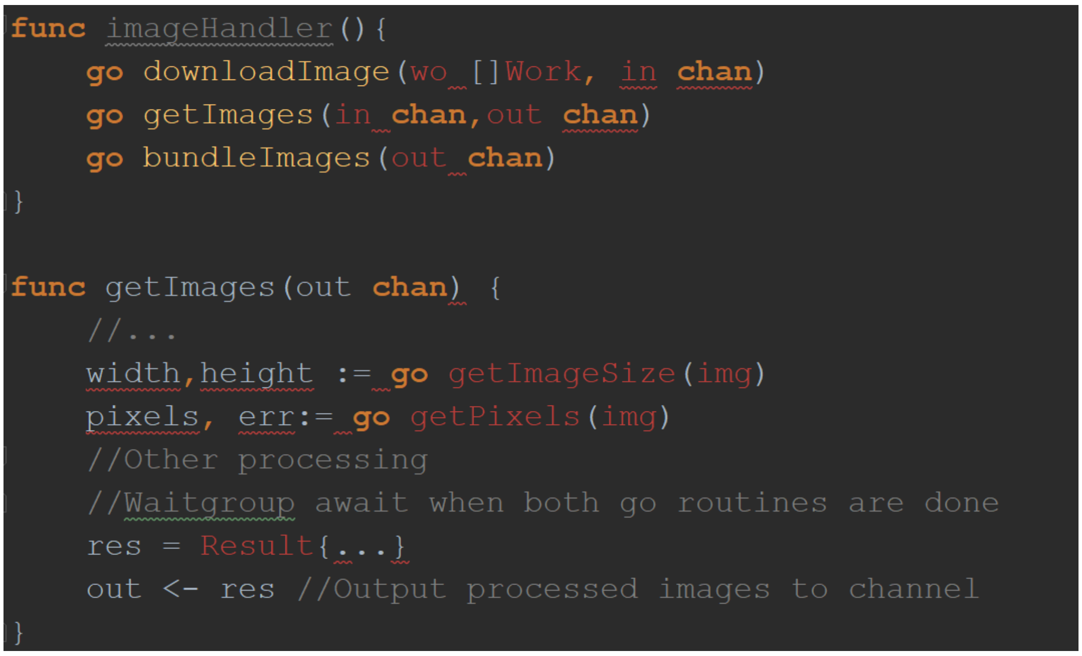
Bottlenecks are found when one is to download images from the URLs, get the images from the body as well as getting the image size and pixels.

Firstly a goroutine is created for each image that is to be downloaded so they can be downloaded concurrently. The downloaded images can be put into an input channel which is then used by the getImage(body) function that is also executed in a goroutine. The getImage() function will take each image from the channel and process them concurrently in their own goroutine. For each image processing, the getImageSize(img) and getPixels(img) may also be executed concurrently since they do not depend on each other. A waitgroup can be utilized here to wait until both functions are returned before the image is further processed.

Each processed image is then put into an output channel which another function can take to bundle all images before returning the result in a response.

In conclusion by propagating some of the work to goroutines one can process the work concurrently and thus increase performance of system.

Some pseudocode to support explaination



```
func imageHandler(){
    go downloadImage(wo []Work, in chan)
    go getImages(in chan,out chan)
    go bundleImages(out chan)
}

func getImages(out chan) {
    //...
    width,height := go getImageSize(img)
    pixels, err:= go getPixels(img)
    //Other processing
    //Waitgroup await when both go routines are done
    res = Result{...}
    out <- res //Output processed images to channel
}
```