Computationally intensive application adaptation for cloud deployment

Martin Myslík

2016

MASTER THESIS

Faculty of Electrical Engineering

Czech Technical University in Prague

Supervisor: Ing. Martin Klíma, Ph.D.

.

Declaration

I hereby declare that I worked out the presented thesis independently and I quoted all the sources used in this thesis in accord with Methodical instructions about ethical principles for writing academic thesis.

Prague, 10th May 2016

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 10. května 2016

Abstract

This work gives an introduction to the biggest cloud vendors on the market and describes a cloud architecture for a computation intensive application. The goal of this application is to search for similarities in large datasets of images. Such task is performed using OpenCV, an open-source computer vision library. The architecture, targeted on AWS infrastructure, is shown with detailed description of its components and focus on specific services offered by the AWS platform. Hypothetical system behaviour with time performance and computational cost is discussed and these assumptions are verified against a reference implementation deployed to the cloud.

Abstrakt

Cílem práce je poskytnout aktuální přehled největších poskytovatelů cloudových služeb a navrhnout architekturu cloudové aplikace pro hledání podobností ve velkých datasetech obrázků. Tato úloha je realizována pomocí OpenCV, open-source knihovny pro computer vision. Architektura aplikace, navržená pro AWS infrastrukturu, je detailně popsána s ohledem na jednotlivé systémové komponenty a specifické služby dostupné na platformě AWS. Hypotetické vlastnosti, především časová náročnost a cena výpočtu, jsou detailně rozebrány a následně ověřeny vlastní implementací nasazené do cloudu.

Acronyms

AWS Amazon Web Services

EC2 Elastic Compute Cloud

ELB Elastic Beanstalk

AMI Amazon Machine Image

RI Reserved Instance

S3 Simple Storage Service

SQS Simple Queue Service

GCE Google Compute Engine

GAE Google App Engine

EMR Elastic MapReduce

Contents

1 Introduction

The rapid development of ICT technologies in the past decades let to the significant emergence of cloud computing. Building large scale applications and processing huge amounts of data is easier than ever before with the source of computation power available in the cloud. Many papers have been written about cloud computing already [19] so this work will only give a brief introduction to the problematic.

There are many available cloud infrastructures and providers offering wide range of cloud services. In the first part of this work, a brief overview of several selected providers will be given, with focus on the core cloud services (namely compute capacity, object storage, database solutions and messaging) and the corresponding pricing models.

Pricing is especially important in this case, since it often affects the application architecture decisions when big data processing is involved. The trend shifted from paying per server instances, where application stacks were managed completely by customers, to the pay-per use model, where users consume services with their own pricing models.

In the second part of this work, a vendor-specific cloud architecture for a computation intensive application, working with large image datasets, will be introduced. Theoretical benchmarks, estimated time and pricing features of the proposed system will be discussed together with additional problems and possible optimizations. The task itself will be introduced later with more details.

Finally, a simplified implementation of the proposed architecture will be shown to prove the functionality and parameters of the application. The main focus will be on several selected implementation problems instead of in-depth components description, since the source code can be found in the attachment to this work.

1.1 Cloud Computing

Cloud computing is a model based on sharing computing resources rather than having personal devices for providing the required functionality and infrastructure. Networks of large groups of servers, typically running low-cost consumer PC technology are grouped by specialized connections to spread data-processing chores across them. This network can then provide computing capacity and other computing resources to the consumers.

The key concept behind provisioning compute capacity in the cloud is to offer an easy way of scaling the application performance. There are two basic models of such scaling:

- **Horizontal scaling:** adding more machines into the pool of resources
- **Vertical scaling:** adding more power (CPU, RAM) to an existing resource

The mechanism and challenges behind cloud instances scalability is a huge topic and will not be explored into detailed in this work. There are several articles [23], [6] in this field.

Many providers set up their own infrastructure in multiple data-centres spread over different regions. Consumers can therefore choose between these locations to guarantee high availability of their hosted applications or ensure cross-region replication of their data.

Cloud infrastructure can be divided into three basic types:

- **public cloud** - set of services delivered by a third party provider via the internet
- **private cloud** - an extension of an enterprise's own datacenter optimised to provide the services and computing capacity locally (infrastructure not shared with third parties)
- **hybrid cloud** - combination of the two

In addition to reducing or eliminating capital expense, many organizations prefer a public cloud solution for its available, on-demand capacity. The hybrid approach is usually applied by storing some sensitive data in a local private datacentre and using the public cloud for everything else.

Cloud Computing is often described as a stack consisting of the following layers:

- **IaaS** - accessing, monitoring, and managing remote datacenter infrastructures
- **PaaS** - computing platform that allows the creation of web applications without the complexity of buying and maintaining the infrastructure underneath it

- **SaaS** - licensing an application to customers either as a service on demand, through a subscription, in a "pay-as-you-go" model or for no charge

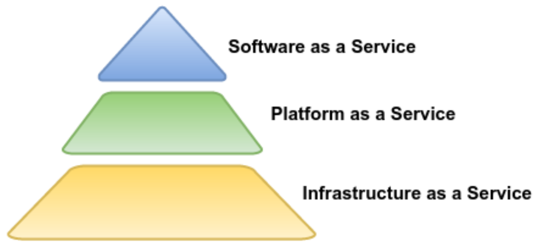

Figure 1: Cloud infrastructure stack

Many related works have been written to cover this problematic [14] [28] so these concepts will not be explored into detail here.

1.2 Virtualization

Virtualization and cloud computing are often confused, since these technologies work closely together. It is, however, important to distinguish between these two terms when referring to 'cloud'.

Simply put, **virtualization** abstracts workloads from hardware [26]. There are several different types of virtualization:

- **Hardware** - virtualization of computers as complete hardware platforms
- **Network** - combining hardware and software network resources into one software-based entity
- **Storage** - grouping the physical storage from multiple storage devices into something that appears as a single storage device, can be divided into block virtualization (abstraction of logical storage from physical storage) and file virtualization (eliminating the dependencies between the data accessed at the file level and the location where the files are physically stored)
- **memory** - virtualization of memory on the application or operating system level
- **data** - for example database virtualization

Cloud computing uses virtualization as its foundation technology, since it allows optimal resource utilization of the physical hardware. Most of the on-demand services offered by various vendors are, in fact, running in a virtual layer and share the physical resources.

2 Cloud Providers Analysis

In this section, selected cloud services providers and their cloud infrastructures will be compared in several categories.

Before reading the overview, it is important to note, that there is a difference between a cloud **provider** and a cloud **infrastructure**. There are some specific cloud infrastructure solutions offered by multiple vendors (providers) and there are also many vendors with their own private infrastructures. The following table[1] shows ten biggest providers with their offered cloud solutions:

Provider	Infrastructure
Amazon	Amazon Web Services
Microsoft	Microsoft Azure
Google	Google Cloud Platform
IBM	IBM Cloud
CenturyLink	CenturyLink
VMWare	vCloud Air
RackSpace	OpenStack
RedHat	OpenStack
Nebula	OpenStack
Joyent	Joyent Triton

Table 1: Cloud providers and their infrastructures

It is not in scope of this work to compare all the existing cloud providers since their number has been growing steadily for the past ten years [27] so only several of them will be discussed in this section. The following vendors has been selected for the overview - Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform and IBM Cloud.

Several categories were selected to be discussed in relation to each provider. These are:

- **Computing capacity** - deploying and managing applications in the cloud
- **Storage** - storing and manipulating objects
- **Databases** - SQL and NoSQL [17] database solutions
- **Messaging/ESBs** - queues and other means of service-to-service communication

[1]Note, that these are just several of the existing cloud providers.

In addition, pricing will be taken into account, since it will play an important role in designing the vendor-specific solution for the system described later in this work.

2.1 Amazon Web Services

AWS was launched in 2006 which makes it the longest operating provider described in this work. It offers wide range of services in 11 geographical regions. Compared to its competitors, AWS offers significantly more compute capacity[2].

Most of AWS services are highly configurable and customizable. This is, on one side, a huge advantage, but on the other hand, setting up a cloud solution on this platform requires slightly more background knowledge and effort.

2.1.1 Elastic Compute Cloud

Amazon EC2 is an IaaS [18] providing access to flexible compute capacity in the cloud. EC2 enables the users to launch, terminate and manage virtual machines via various tools including graphical web interface or AWS command line interpreter (AWS CLI). These virtual machines are called "instances" and the user is able to boot an Amazon Machine Image (AMI)[3] on them.

AWS Marketplace

Not only can users choose from number of predefined AMIs provided by Amazon, but there is also a possibility to buy one from AWS Marketplace[4]. Customers can trade reserved instances, sell custom AMIs or offer software running on EC2. Moreover, it is possible to list the service offering programmatically and therefore automate the process of buying and selling the software or reservations.

At the moment, Amazon offers 9 instance families and 41 instance types types[5] in a wide range of sizes to match specific needs, including GPU or CPU heavy applications. Several analysis have been done, concerning the EC2, indicating varying, yet stable, performance,of different instance types [7].

[2] `https://www.srgresearch.com/articles/aws-market-share-reaches-five-year-high-despite-microsoft-growth-surge`
[3] `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html`
[4] `https://aws.amazon.com/marketplace`
[5] `https://aws.amazon.com/ec2/instance-types/`

Instances running in EC2 are always priced for the total number of hours used (rounded up) according to the following three models:

- **on demand** – no upfront costs
- **reserved** – customers can reserve instances for 1 or 3 years (or buy existing reservation from the marketplace) with some upfront cost based on the utilization
- **spot** – customers bid on spare instances

By using reserved or spot instances, users can significantly reduce their operational costs. Since cost-optimization of AWS compute capacity is more advanced (and also more complex) compared to the other providers discussed in this work, the key concepts will be shortly described below.

Reserved instances

Although called reserved instances, these resources are not instances in the classic sense. They are just reservations of computing capacity for specific instance configurations. When purchasing a reserved instance, Amazon guarantees that it will be possible to run your instance in a selected region/zone with the specified parameters (like platform, instance type etc.). Moreover, Amazon guarantees lower prices when using reserved instances compared to running the instances as on-demand.

In order to describe what is meant by this concept, imagine instances being cars and reserved instances highly customized parking spaces. You can either park your car on a public parking lot and pay an hourly price or use your prepaid parking space to save money. Also note, that reserved instances are being charged no matter whether they are being used or not.

This table shows an example pricing of several EC2 instance types with reserved instance plans:

Instance	vCPU	Mem (GiB)	On demand	RI no upront	RI all upfront
t2.micro	1	1	\$0.013	\$0.009	\$0.0086
t2.small	1	2	\$0.03	\$0.022	\$0.0201
c4.large	2	3.75	\$0.134	\$0.100	\$0.0862
r3.large	2	15	\$0.200	\$0.146	\$0.123

Table 2: Example EC2 pricing (with One year RI terms) - US-east, Amazon Linux

Spot instances

Spot instances are, simply put, unused EC2 instances. The price is often significantly lower than the price for reserved instance and fluctuates depending on the current supply and demand on the market. An interesting study [1] has been performed, however, indicating that the prices of spot instances might not be entirely market-driven. The disadvantage of using spot instances is that these instances can be terminated by Amazon (there is a notice 10 minutes before the termination) and might not be launched immediately (since AWS waits for such instance to be available). These are an ideal choice for applications that do not need to be running without interruptions.

2.1.2 Elastic Beanstalk

Elastic Beanstalk is a service which reduces the complexity of deploying applications to AWS. Users can upload the application and let Elastic Beanstalk handle the version control, capacity provisioning, scaling and application health monitoring.

Elastic Load Balancing automatically distributes and balances the incoming application traffic among all the running instances. The load balancer sends requests to the Amazon EC2 instances that are running the deployed application.

The default behaviour is to distribute the load across instances running in the same availability zone, but **cross-zone** load-balancing can be enabled to use all the instances, regardless of which zone they are in. An important thing to note is, that ELB needs some time to launch a new instance when scaling the application during a spike in demand. This is called **pre-warming** and it might take a couple of minutes before the instance becomes available. This might cause problems when handling very large and short spikes.

The applications can be deployed to ELB either as a ”website” (with public interface) or as a ”worker” (with only one endpoint linked to a queue).

Elastic Beanstalk is not a subject to any charges. Users only pay for the EC2 instances managed by Beanstalk. There is a whole book [24] dedicated solely to ELB for those interested in in-depth description of this service.

2.1.3 Simple Storage Service

Amazon Simple Storage Service provides a scalable object storage accessible via web interface or API. It offers three storage classes designed for different scenarios:

- **Standard** - general purpose storage
- **Standard-IA** (infrequent access) - less frequently accessed data
- **Glacier** - long-term archive

Objects are stored in *buckets* which can be configured both with access rights and additional functionality (automatic archival, life span etc.). Buckets can be used not only for file storage but also for a website hosting or content distribution.

Data size	Standard	Standard-IA	Glacier
First TB / month	\$0.0300	\$0.0125	\$0.007
Over 500 TB / month	\$0.0275	\$0.0125	\$0.007

Table 3: S3 storage pricing, US, price per GB

Request	Standard	Standard-IA
Put, Copy, Post	\$0.005 per 1000 requests	\$0.01 per 1000 requests
Get and other requests	\$0.004 per 10000 requests	\$0.01 per 10000 requests
Delete	free	\$0.01 per GB

Table 4: S3 request pricing

S3 Glacier is charged differently since it only uses archive and restore operations. These are significantly cheaper than the other two options, but it takes several hours to access the archived data so it is not meant for standard usage.

2.1.4 DynamoDB

Amazon DynamoDB is Amazon's scalable NoSQL database. Data entries saved in DynamoDB are accessible by their hash keys but can also be queried using secondary indexes.

Usage of DynamoDB is charged for both the stored data size and for provisioned throughput capacity. Throughput capacity is, simply put, a guaranteed throughput for a given table which enables the user to specify how many requests should be achieved per second.

Operation	Price
Storage (monthly)	$0.25 per GB
Write throughput (hourly)	$0.0065 per 10 units
Read throughput (hourly)	$0.0065 per 50 units

Table 5: DynamoDB pricing

The provisioned throughput can be scaled up an down for every DynamoDB table. In other words, if an application requires to process large amount of data in the database, the throughput can be increased just during the processing and lowered afterwards to decrease the costs.

One limitation of DynamoDB is the single item size which cannot exceed 64KB, athough it can be avoided using techniques like paging. This is a tradeoff made by Amazon to ensure the guaranteed latencies and data replication of this service.

Since DynamoDB is more complex than a simple key-value store, it requires at least some basic level of understanding before users can take the full advantage of all its capabilities. There are several puplications [25] [9] dedicated to DynamoDB providing a detailed description, best practices and examples of usage of this service.

2.1.5 Relational Database Service

Amazon RDS a relational database solution in the cloud. It supports six database engines: Amazon Aurora, Oracle, Microsoft SQL server, PostgresSQL, MySQL and MariaDB.

RDS is priced, similarly to EC2, for the total number of hours that the database instances were running. There are various instance types optimised for different tasks (standard, memory optimised, micro etc.). The following table shows an example pricing of several selected instance types. Like EC2, users can pay for reserved database instances to decrease the costs for a long term usage.

The following table illustrates the pricing for MySQL instances in the EU region:

Instance	vCPU	Mem (GiB)	On demand	RI no upront	RI all upfront
db.t2.micro	1	1	$0.018	$0.014	$0.012
db.m4.large	2	8	$0.193	$0.146	$0.023
db.r3.large	2	15	$0.265	$0.199	$0.166

Table 6: Example RDS pricing (with One year RI terms), US-east

2.1.6 Simple Queue Service

Amazon SQS is a scalable message queuing service. It is possible to transmit data of any size and any throughput with this service regardless of the fact whether the consuming services are available at the time.

There are no hourly costs but users pay the data transfer:

Data transfer	Price
First GB / month	free
Up to 10 TB / month	$0.090 per GB
More than 10 TB / month	$0.050 - 0.0850 per GB

Table 7: SQS pricing

2.2 Microsoft Azure

Azure, announced in 2008 and officially released in 2010, is a cloud computing platform created by Microsoft. It provides both PaaS and IaaS services through a global network and Microsoft-hosted data-centres.

Azure provides a strong support for hybrid cloud architecture, in contrast with Amazon which is only beginning to adopt this concept. The hybrid model allows certain parts of an application to remain on on-premises hardware and the interfaces with other parts to be hosted in the cloud.

Another important feature of Azure is its high compatibility with other Microsoft products, making it a solid choice for customers using this stack. A detailed description of Azure services and their underlying infrastructure can be read in this [12] publication.

2.2.1 Virtual Machines

Virtual machines has been available since 2012 and currently offers 4 instance families with 33 instance types in a variety of sizes. Customers can create virtual machines, of which they have complete control, to run in Microsoft's data-centres.

As for pricing, Microsoft charges customers by the total number of minutes used with the possibility of short-term commitments with discounts[6]. In the past, subscribers were able to get 20-32% discount with 6 to 12 months commitments but Microsoft changed the pricing policy in 2015. At the moment, there is a global discount 5% for all Azure services when subscribing for 12 months[7].

Microsoft offers running the instances in either basic or standard tier[8]:

- **Basic** - development workloads, no auto-scaling and load balancing
- **Standard** - all virtual machine configurations and features

Instance	vCPU	Mem (GiB)	Basic on-demand	Standard on-demand
A1	1	1.75	$0.047	$0.06
D1	1	3.5	-	$0.077
D2	2	7	-	$0.154
D11v2	2	14	-	$0.224

Table 8: Example Azure Virtual machines pricing, US-central, Linux

2.2.2 Cloud Services

Cloud Services is a PaaS environment for deploying scalable applications to the cloud. Similarly to Amazon's Elastic Beanstalk, the applications have **roles** indicating their purpose and interface accessibility: "Web Role" for a public internet-exposed application and "Worker Role" for a private processing engine.

Microsoft currently supports four programming languages to be used with Cloud Services: Java, Python, Node.js and .NET.

2.2.3 Blob Storage

BLOB (Binary Large Objects) Storage can be used to store and retrieve binary files in Azure cloud. All data stored with this service are automatically replicated and users can choose from 19 regions to host their files. All objects (blobs) can be accessed by a unique path.

There are two kinds of blobs with the type specified during the blob creation:

[6] `https://azure.microsoft.com/en-us/offers/commitment-plans/`
[7] ttps://azure.microsoft.com/en-us/offers/ms-azr-0026p/
[8] The basic tier is available only for selected instance families

- **Block blob** - 64MB blocks can be uploaded in one operation. Larger blobs (up to 200GB) can be uploaded in smaller parts and then put together as one file once the upload is finished.
- **Page blob** - These blobs can be up to 1TB in size and consist of 512-byte pages.

When creating the blob, users can specify the replication strategy - locally redundant storage (LRS), zone redundant storage (ZRS) and geographically redundant storage (GRS). The pricing is different of each of these models:

Data size	**LRS**	**ZRS**	**GRS**
First TB /month	$0.024 per GB	$0.03 per GB	$0.048 per GB
Next 49TB /month	$0.0236 per GB	$0.0295 per GB	$0.0472 per GB
Next 450TB /month	$0.0232 per GB	$0.029 per GB	$0.0464 per GB

Table 9: Block blob storage pricing

Data size	**LRS**	**ZRS**	**GRS**
First TB /month	$0.05 per GB	$0.095 per GB	$0.12 per GB
Next 49TB /month	$0.05 per GB	$0.08 per GB	$0.1 per GB
Next 450TB /month	$0.05 per GB	$0.07 per GB	$0.09 per GB

Table 10: Page blob storage pricing

Read and write operations are both charged $0.0036 for 100k operations.

2.2.4 DocumentDB

DocumentDB is Microsoft's NoSQL cloud database. Compared to Amazon DynamoDB, it is a fully functional document-oriented database with ACID transactions (within a collection of stored procedures) and a built-in JSON and Javascript support. The throughput is controlled by the system and there is no need to specify it beforehand.

One of the downsides of DocumentDB is a very limited range of supported programming languages (.NET, Python, Javascript[9]).

Microsoft charges customers based on the number of collections contained in a database account. Collections are billed separately on hourly basis and the pricing depends on the specified performance class:

[9] Third party plugins for other programming languages exist, though.

Class	SSD storage	Request units	Price
S1	10 GB	250 / second	\$0.034
S2	10 GB	1000 / second	\$0.067
S3	10 GB	2500 / second	\$0.134

Table 11: DocumentDB pricing

2.2.5 SQL Database

Azure SQL Database is Microsoft's database-as-a-service. Though somehow similar to Amazon RDS, there are several key differences to note between these two services. First of all, unlike Amazon, which enables users to have dedicated hardware resources, Microsoft SQL Database shares these resources between all users. This is called **Elastic database model** and is used for managing the collective performance of the pool rather than single databases. The databases in the pool automatically scale up and down according to the demand.

SQL Database doesn't offer backups and restoration automatically and this functionality has to be manually configured. Lastly, this service only supports MySQL at the moment.

Microsoft pricing is based on the total time of running the elastic database on hourly basis. There are three tiers distinguished by the overall performance:

Tier-eDTUs/pool	Storage	DBs/pool	eDTUs/DB	Price
Basic-100	10 GB	200	5	\$0.20
Basic-200	20 GB	200	5	\$0.40
Standard-100	100 GB	200	100	\$0.30
Standard-200	200 GB	200	100	\$0.60
Premium-125	63 GB	50	125	\$0.937
Standard-250	125 GB	50	250	\$1.88

Table 12: SQL Database example pricing

DTU and eDTU

DTU stands for "Database Transaction Unit" and it is Microsoft's unit representing the relative power of database based on the real-world measure of database transaction. In other words, DTUs indicate how many transactions per second can be performed over a given database system. Similarly to DTU, eDTU is the same unit applied to an elastic database pool.

2.2.6 Queue Service

Two types of queue mechanisms are offered by Microsoft's cloud platform[10]: Azure Queues and Service Bus Queues.

Azure queues have a REST interface supporting Get/Put/Peek operations and are used for point-to-point connections between services. **Service Bus queues**, however, offer publish/subscribe mechanism as well as topics/subscriptions.

The following table compares these two queue services in several categories:

Criteria	Azure queues	Service Bus queues
Ordering	Not guaranteed	Yes - FIFO
Delivery	At least once	At least once, at most once
Transactions	No	Yes
Batches	No	Yes
Automatic dead-lettering	No	Yes
Purge queue	Yes	No
Duplicate detection	No	Yes
Message groups	No	Yes
Maximum size	200 TB	80 GB
Maximum message size	64 KB	256 KB
Message TTL	7 days	Unlimited

Table 13: Queues comparison

The queue pricing is based on pay-per-use model and due to an extensive list of options and configurations, it will not be discussed here. These pricing details can be found on the official Microsoft website[11].

2.3 Google Cloud Platform

Google Cloud Platform was first launched in 2008 as a preview with Google Compute Engine being the first available service. The platform offers a set of modular services providing a complete infrastructure for cloud-based systems. It is the same infrastructure that Google uses internally for its services like Google Search. There are currently four regions supported by the platform.

[10] Although it is possible to install and run third party queues in Azure.
[11] `https://azure.microsoft.com/en-us/pricing/details/service-bus/`

Compared to Amazon, Google provides smaller number of services and regions. These services often lack sophisticated configuration possibilities but their setup is easier and does not require an extent background knowledge. Google focuses mainly on providing PaaS rather than IaaS.

Additionally, Google provides unique solutions like BigTable or BigQuery, which makes it an excellent choice for big data processing.

2.3.1 Compute Engine

Google Compute Engine (GCE) is an IaaS enabling the users to configure and run virtual machine instances in the cloud. GCE currently supports 4 instance families with 18 instance types in total. It was first launched in 2012 as a preview but it became officially available in 2013.

Unlike AWS, GCE charges users by the number of minutes used (with 10 minutes being the minimum). It only provides on-demand pricing with sustained use discounts[12], making instances running for a significant portion of the month, eligible for discount. Basically, instances running during the first 25% of the month are charged more than instances running for 25% to 50% and so on. When running instances for the whole month, they cost up to 20% less than the on demand prices.

Moreover, GCE allows user to run instance in a **preemptible** mode. However, these instances might be terminated (similarly to AWS spot instances) if GCE it requires access to the resources for other tasks. The cost for running such instances is significantly lower than the on demand price. Sustained use discounts are not applied to such instances.

Instance	vCPU	Mem (GiB)	On demand	Full sustained use	Preemptible
n1-standard-1	1	3.75	\$0.050	\$0.035	\$0.015
n1-standard-2	2	7.5	\$0.100	\$0.070	\$0.030
n1-highcpu-2	2	1.8	\$0.076	\$0.053	\$0.020
n1-highmem-2	2	13	\$0.126	\$0.088	\$0.035

Table 14: Example GCE pricing, US, Linux

2.3.2 App Engine

Google App Engine (GAE), first launched in 2008, is a PaaS for managing applications deployed to cloud. The auto-scaling and load-balancing mechanism is available by default without the need

[12] `https://cloud.google.com/compute/pricing`

of additional service configuration. GAE's strong side is the compatibility and integration with existing development tools like Eclipse, Maven, Jenkins or Git. Another important feature of App Engine is the **Security Scanner**, which can be configured to automatically scan deployed applications for vulnerabilities.

It is possible to use GAE without GCE (although user can combine these two services easily). That reduces the cloud management to minimum since App Engine can handle everything automatically. The downside GAE is the limited number of operating system and platform choices (PHP, Python, Jana and Go are supported).

With GAE, users only pay for what is used. This applies for the compute capacity, datastore calls etc[13]. The pricing of App Engine will not be described into details here since the focus of this work is to provide an overview of large scale application solutions which are likely to use GCE with other resources configured manually instead.

2.3.3 Cloud Storage

Cloud Storage is a highly available object storage which comes in three product options sharing the same API (similarly to Amazon S3):

- **Standard** - highest level of durability and availability
- **DRA** - lower availability
- **Nearline** - storage for archiving, backup and disaster recovery

Another concept shared with Amazon S3 is the use of **buckets**. Storage options are specified on a bucket level with **Standard** storage being the default one. Each bucket can be hosted in different location which is set during its creation.

Option	Availability	Price
Standard	99.9% — ms	$0.026 per GB
DRA	99% — ms	$0.02 per GB
Nearline	99% — sec	$0.01 per GB

Table 15: Cloud Storage pricing

[13] `https://cloud.google.com/appengine/pricing`

In addition to these operations, operations with objects and buckets are charged as well:

Operation	Price
Insert, update, patch, list, copy, delete	$0.1 per 10000 operations
Get, access control, notifications	$0.01 per 10000 operations

Table 16: Cloud Storage operation pricing

Additional charges for Egress might be applied as well[14].

2.3.4 Datastore

Google Datastore is a scalable NoSQL database with automatic sharding and replication. It is quite similar to Amazon DynamoDB - both options offer schema-less table structures, list properties and simple index keys. Unlike Google Datastore, DynamoDB is able to perform a full table scan (although it is highly discouraged due to low efficiency).

However, there are some important differences between Google Datastore and Amazon DynamoDB. Unlike DynamoDB, Datastore has a built-in transaction system and an entity hierarchy support. It is not necessary to specify the throughput capacity like in DynamoDB since this is handled automatically in Datastore (which implies simpler usage but also lower control over large data manipulation).

Pricing is applied to data storage and read/write operations:

Operation	Price
Storage	$0.018 per GB per month
Read, write	$0.06 per 100000 operations

Table 17: Cloud Storage operation pricing

2.3.5 Cloud SQL

Google Cloud SQL is a fully-managed database service. Data encryption is applied automatically on all data saved in the Cloud SQL and all the data are replicated over multiple regions and backed up routinely.

There are two pricing plans for Cloud SQL[15]

[14] `https://cloud.google.com/storage/pricing`
[15] `https://cloud.google.com/sql/pricing`

- **Packages Billing Plan** - ideal for sustained database usage
- **Per Use Billing Plan** - ideal for non-frequent usage (charged by hour)

Similarly to Amazon RDS, several instance types with different RAM and storage options can be selected to host the database:

Instance	RAM	Price / day	Price / hour
D0	0.125 GB	\$0.36	\$0.025
D1	0.5 GB	\$1.46	\$0.10
D8	4 GB	\$8.78	\$0.58

Table 18: Example Cloud SQL pricing

2.3.6 Cloud Pub/Sub

Messaging in the Google platform is handled by Cloud Pub/Sub. This service is global by design - it remains available even when some zones or regions aren't.

Google uses its private global fibre network to transfer the messages. This is an important difference between Google and its competitors, who often use the public network.

Operations	Price/million per month
First 250M	\$0.40
Next 500M	\$0.20
Next 1000M	\$0.10
More than 1750M	\$0.05

Table 19: Pub/Sub pricing

2.4 IBM Cloud

IBM offers a complete set of cloud services through public, private and hybrid delivery models. These offerings used to be marketed under IBM SmartCloud brand since the first launch of enterprise cloud service in 2011. In 2013, IBM bought SoftLayer (server, hosting and cloud computing provider) which meant a huge improvement of the IaaS capabilities.

Private and hybrid cloud solutions are strong sides of IBM cloud, since public cloud has not been the main area of interest until recently.

2.4.1 SoftLayer Virtual Servers

IBM offers its IaaS through SoftLayer Virtual Servers. Unlike its competitors, SoftLayer does not offer specific instance types and instance families, but the individual instance configurations (CPU, memory, disks etc.) can be specified by the user before launching.

There are six operating systems in multiple flavours supported[16]: CentOS, CoreOS, Debian, Microsoft, Redhat and Ubuntu.

Before jumping to the pricing, it is important to note that SoftLayer Virtual machines is fundamentally different to, for example, Amazon EC2. EC2 is running on a virtualization layer, which searches for a free node with requested resource whenever someone orders to launch an instance. This process is quick and automated and gives the customer a complete control over all the resources on the virtual layer. On the other, SoftLayer gives the customers the same control on the infrastructure layer. In other words, SoftLayer deploys bare metal servers in a similar same way AWS deploys public cloud servers. Another difference to note is the lack of regions and availability zones in SoftLayer. Customers select a specific data centre to host their instances and are able to expand their cloud infrastructure to different data centres which are interconnected via a global private network.

There are two pricing models available with the Virtual Servers service: hourly on-demand and monthly pricing[17].

vCPU	Mem (GiB)	Hourly	Monthly
1	1	$0.038	$25
1	2	$0.053	$35
2	4	$0.105	$70
2	12	$0.208	$138

Table 20: Example Virtual Servers pricing, US-Washington-DC, Linux

[16]It is also possible to launch a custom OS with the instance.
[17]Public node prices are listed here. Private nodes are more expensive:

2.4.2 Bluemix

IBM Bluemix, officially launched in 2014, is a PaaS enabling users to create, deploy and manage applications in the cloud. Bluemix platform is based on CloudFoundry[18] and runs on the SoftLayer infrastructure. Several programming languages are supported including Java, Node.js, Go, PHP, Python or Ruby. Bluemix consists of all the services related to managing the cloud applications, including storage, databases, messaging, monitoring etc.

Unlike AWS, Bluemix enables hosting the applications as on-premise and it also offers a connection to IBM's Watson - a cognitive machine learning system.

Similarly to GAE or ELB, customers are charged for various services consumed by the default application. Therefore, the detailed pricing is not going to be discussed in this section[19].

2.4.3 Object Storage

Object Storage is used for unstructured data storage and is currently offered as a beta release. Apart from the standard features like access control or service dashboards, Object Storage adopted the OpenStack Swift API and SKDs for accessing the stored objects programmatically.

As for pricing, there are currently two plans available:

- **Free** - Single instance with 5GB limit
- **Beta** - Multiple instances with scalable storage and pay-per-use pricing model

Unfortunately, by the time of writing this work, there was no available pricing details for Object Storage since it has not come out of beta yet.

2.4.4 Cloudant NoSQL DB

Cloudant is a schema-free NoSQL database-as-a-service based on Apache CouchDB[20]. Data are accessible via Http REST interface in JSON format and records can be treated independently or in

[18] `https://www.cloudfoundry.org/`
[19] IBM has a detailed price calculator on its website: `https://console.ng.bluemix.net/pricing/`
[20] `http://couchdb.apache.org/`

bulks. Attaching files to documents is supported as well.

One of the strong features of Cloudant is a full-text search based on MapReduce and Apache Lucene[21].

Cloudant offers 20GB of free storage. The additional data are charged according to the following pricing plan:

Operation	Price
1GB storage	\$1.00
1000 read calls	\$0.03
1000 write/update/delete calls	\$0.15

Table 21: IBM Cloudant pricing, US

2.4.5 SQL Database

This service provides an on-demand relational database powered by IBM's DB2 engine[22]. All data is automatically encrypted and some advanced security features, like data masking, are available as well. Moreover, automatic backup, monitoring and daily reports are available by default.

SQL Database offers two pricing plans: a free one with limited storage and a premium one. Customers are charged for the number of instances on monthly basis.

Plan	Storage	Connections	Price
Free	100MB/instance	10	-
Premium	500GB/instance	100	\$400/instance

Table 22: IBM SQL Database pricing, US

Apart from SQL Database, there are additional services providing various database functionality integrated in Bluemix:

- **IBM DB2 on cloud** - fully private, dedicated instances
- **Elephant SQL** - third party multi-tenant Postgres database, can also be hosted on AWS, GCE and Azure

[21]https://lucene.apache.org/
[22]http://www-01.ibm.com/software/data/db2/

- **Time Series Database** - specialized time series database with both SQL and NoSQL functionality

- **ClearDB** - third party MySQL database, can be also hosted on AWS, GCE and Azure

2.4.6 MQ Light

Messaging between applications is handled by MQ Light messaging service. MQ light allows both point-to-point communication and publish-subscribe model with topics. As for the supported programming languages, users can choose between Ruby, Node.js, Java and Python, PHP, Perl and C.

Pricing is based simply on the total number of messages sent via this service:

Messages	Price
First 100000	-
1 million	$5.0

Table 23: IBM MQ Light pricing, US

MQ Light, however, is not the only messaging option integrated in Bluemix. Besides various custom queue engines, the following options are also available:

- **Message Hub** - distributed message bus based on Apache Kafka[23]

- **IBM Push Notifications** - push notifications for Android and iOS devices

2.5 Overall Summary

This section will briefly summarize all the discussed cloud providers with focus on their main areas of expertise. A price comparison of selected services will be given as well, although not every category will be evaluated since every provider uses slightly different pricing model and offers different set of features with every service, which would result in unfair comparison.

One of the biggest differences between the discussed cloud vendors is the number of available services. Amazon, being the first company to launch their public cloud infrastructure, has the biggest number of them, making it's platform the most complete on the market at the moment. The other three competitors are, however, introducing new services on regular basis and cover all the discussed

[23] `http://kafka.apache.org/`

areas of cloud computing as well.

On the other hand, while AWS focus is on the IaaS offerings, Google, Azure and IBM have very mature PaaS solutions, making the development of applications on their platforms slightly easier.

Another huge aspect of building a cloud stack are the requirements of the customers. Azure, for instance, offers the best integration capabilities with other Microsoft services and also seems to have the best support for hybrid cloud solutions. AWS let users configure almost all aspects of their cloud solution, although more experience and time is required to set-up the application. GCE is specialized in big data processing, with Google having lots of experience in this topic. And lastly, IBM cloud offers API to many cognitive and AI services, like Watson, giving it a unique appeal[24]. Moreover, with its SoftLayer Virtual Servers offering, it enables customers to host their services on bare metal machines, rather on a virtual layer like the other providers[25].

2.5.1 Pricing Comparison

One of the crucial parts of a cloud system are the virtual instances that host the application and perform the computation tasks. These will be compared in terms of price in the following three categories: general usage, high CPU usage and high memory usage. Unfortunately, it is not possible to find a perfect mapping between instance types across multiple cloud providers so the results must be treated with caution.

The following instances were selected for the comparison:

[24] This does not mean that some of the other providers does not have such services in their stacks. Their solutions are not as mature, though.

[25] This might change in the near future since Amazon just announced a new "dedicated machine" feature of its EC2 service

Instance	vCPU	Memory	Memory
AWS t2.small	1	2	$0.030
Azure A1	1	1.75	$0.060
GCE n1-standard-1	1	3.75	$0.050
IBM SoftLayer-1-2[26]	1	2	$0.053
AWS c4.large	2	3.75	$0.134
Aure D2	2	7	$0.154
GCE n1-highcpu-2	1	1.8	$0.076
IBM SoftLayer-2-4	2	4	$0.105
AWS r3.large	2	15	$0.200
Aure D11v2	2	14	$0.224
GCE n1-highmem-2	12	13	$0.126
IBM SoftLayer-2-12	2	12	$0.208

Table 24: IaaS instances pricing table

The price reflected in this table is the on-demand price for Linux instances in the US region.

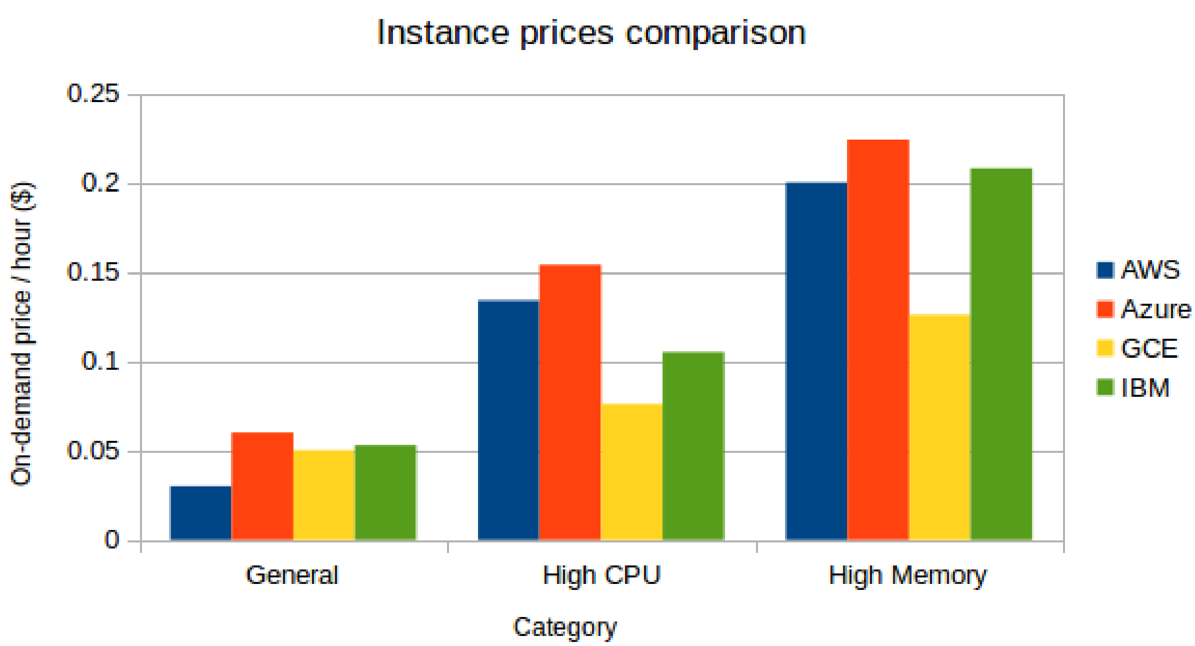

Figure 2: Instance pricing comparison

While all the compared providers offer similar instance types, the situation is completely different with object storage offering. IBM Cloud does not have this service in its portfolio yet, since its Object Storage is only available as a beta version at the moment. Azure offers different kinds of its Blob storage in multiple flavours (LRS, ZRS, GRS) and AWS has different types of its S3 storage as well.

Only the basic storage types are shown in the following table, all of them with prices in the US region for the first TB of saved data:

Service	1 GB	100k reads	100k writes
AWS S3 (Standard)	\$0.03	\$0.04	\$0.5
Azure Block Blob Storage (LRS)	\$0.024	\$0.0036	\$0.0036
GCP Cloud Storage (Standard)	\$0.026	\$0.1	\$1

Table 25: Object storage pricing comparison

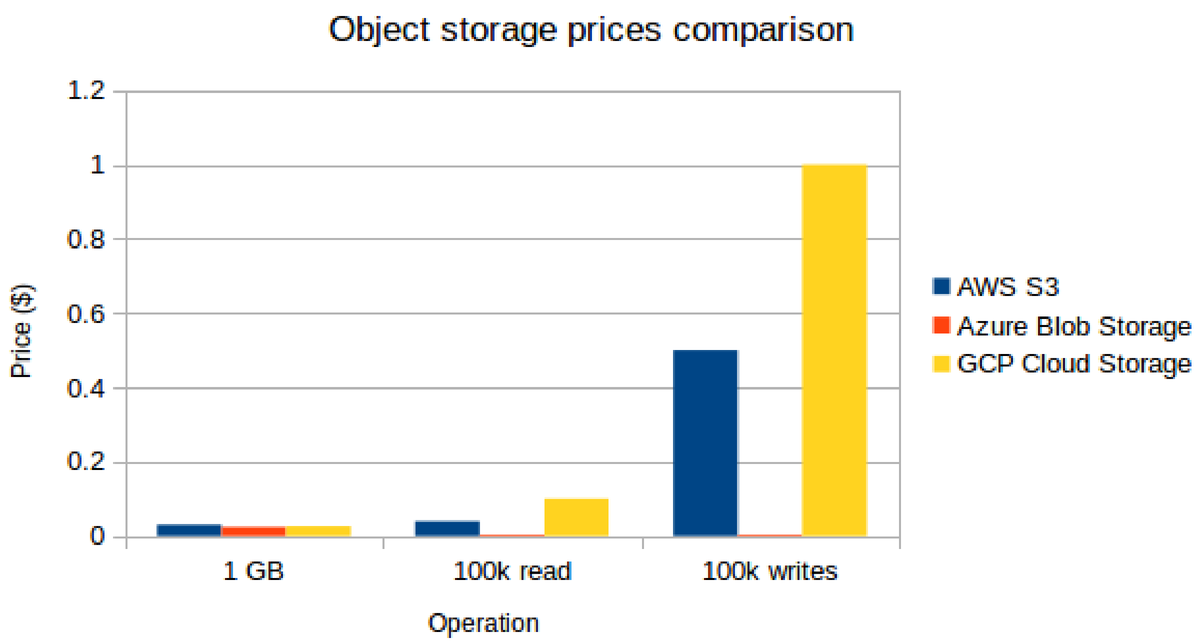

Figure 3: Instance pricing comparison

Google seems to have significantly higher prices for PUT requests so it might not be an ideal choice for heave-write applications. On the other hand, it does not charge for any transport across its multi-region locations (transport of an object from US Cloud Storage to EU compute instance is free). Amazon and Azure charge for any data transfer between regions or data centers.

SQL and NoSQL services comparison will not be discussed here since the pricing models of the individual providers are too different and it would, hence, result in unfair outcome.

3 Problem Description

As mentioned in the introduction, the purpose of this work is to design a cloud architecture for an application handling large datasets of images and ensure its scalability and robustness in terms of processing large workloads.

Firstly, the system description and requirements will be introduced. A description of worker tasks will be given, with emphasis on input and output parameters, workload classification and time performance.

Secondly, an architecture for one selected cloud provider will be described in the following chapters together with time and price estimations for big workloads.

Lastly, additional possible optimizations over the discussed solutions will be explored.

3.1 Task Description

The main task of the system is to identify duplicates and similarities in a large collection of images. There will be two operations performed over the dataset:

- **1-N comparison** - compare one image with all the images in the dataset
- **N-N comparison** - compare all images in the dataset against each other

The results of the comparison must be persisted. The best matches are then presented to the user.

With growing number of images in the dataset it takes considerable amount of computational resources and time, especially with the N-N comparison, to perform the tasks, so the system must be able to split the workload over multiple worker instances and minimize the time overhead caused by traffic and database operations.

Moreover, careful selection of cloud services is crucial in terms of pricing since some of the tasks performed by this system might run for couple of days.

3.1.1 Image Pre-processing

Firstly, an image is uploaded to the system with tags specifying the image content. This image must be saved somewhere as a file in order to make the future retrieval possible.

Once the image is saved, several algorithms are executed to compute sets of **features** describing the image. **OpenCV**[27] [5] library is used for all the image processing operations. Each algorithm produces its own set of feature vectors.

It is difficult to define the meaning of the term *image feature* in this context but we can image it to be a specific pattern which can be easily tracked and compared. Once OpenCV finds these features, it needs to convert them into some format which could be used for further processing. This phase is called *feature description* and it turns the extracted features into vector representation.

There are number of different algorithms that could be used for the features and image descriptors extraction[28]:

- **ORB (Oriented FAST and Rotated BRIEF)** - local feature detector based on the FAST keypoint detector and the visual descriptor BRIEF [22], published as open-source
- **KAZE** - 2D feature detection and description algorithm in non-linear scale spaces, published as open-source [3]
- **AKAZE** - accelerated version of KAZE with increased performance and precision [4]
- **SIFT (Scale-invariant feature transform)** - local feature descriptor invariant to uniform scaling and orientation [13], licensed and patented by University of British Columbia
- **SURF (Speeded up robust features)** - local feature detector inspired by SIFT, licensed and patented in the United States

Finally, **image descriptors** are computed based on the extracted features. These descriptors are then used for the image comparison. For the sake of simplicity, these entities will be called just 'feature vectors' in this work.

[27] http://opencv.org/

[28] Only several selected algorithms are listed here - these are is a wide range of approaches to image feature extraction.

The problematic of content-based image retrieval will not be described into detail here, since it is not important for the behaviour of the proposed system. There are, however, many other works discussing this topic [8] [11].

After performing these operations, the image is ready to be compared with other image representations in the system.

3.1.2 Images Comparison

Two images can be compared against each other using the pre-computed feature vectors. This is called **feature matching**. Only vectors generated by the same algorithm can be used in one such operation. The functionality for matching the feature vectors is also part of OpenCV.

The most simple way of comparing two sets of feature vectors representing the compared images is **Brute-force matching**. In this case, the descriptor (vector) of one feature from the first set is compared with all the features from the second set. The closest match, using some distance function, is returned. This has to be done for all features in the first set.

Another approach is to use a **k-nearest-neighbours matcher**. This matcher returns k matches instead of just one. This enables us to perform a ratio test and remove outliers. For example, if the distances of one descriptor from the first set are similar to the k descriptors in the second set[29], it means that there are repetitive patterns in the image and such descriptors are not reliable (and, thus, should be removed).

K-nn matching will be used in the system described in the following sections, since it provides more accuracy for comparing the images [16].

As a result of the comparison, a value is must be returned, representing the similarity of the compared images. Number of good matches (matches with distance under some specified threshold) can be used to generate such value. This approach will be used to score the images similarity later.

[29] Usually, $k = 2$ is used

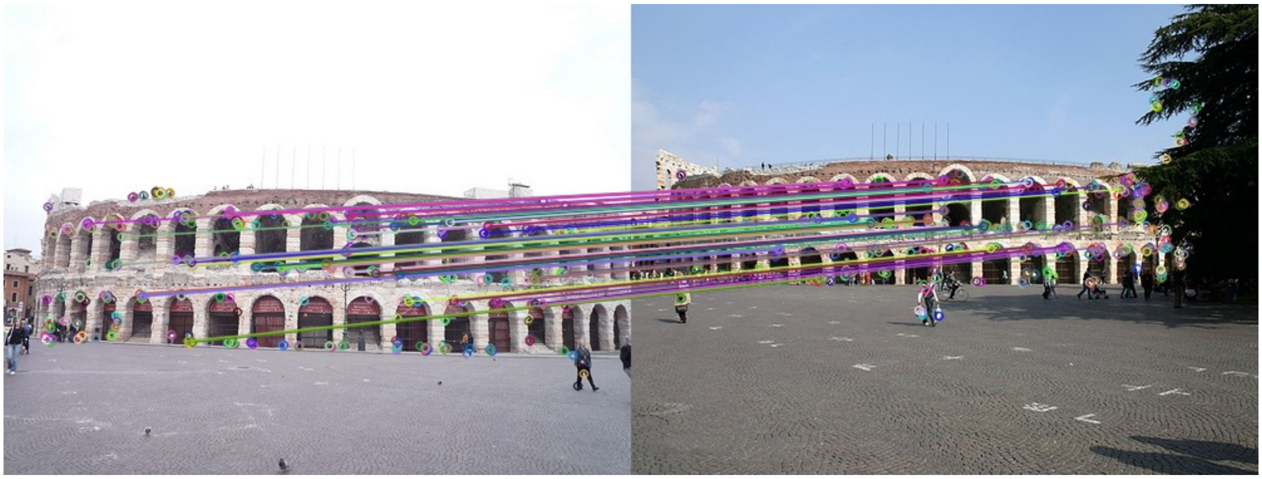

Figure 4: Example of identified matches using ORB detector and knn matcher

Figure 5: Example of identified matches using AKAZE detector and knn matcher

The following workflow is used for the image comparison:

- Load feature vectors (image descriptors) of the two images being compared
- Use OpenCV matcher to find matches
- Discard all matches below a threshold to reduce the number of incorrect results
- Compute the result based on the number of remaining matches and their distance

It is important to note, however, that using features for comparing images might not be accurate in some cases. Feature count and quality can vary with picture size and content. This can make it difficult to compare images in global scale. The overall quality of the results can be increased, for example, by combining the feature extraction with image histogram analysis[30] [21].

[30]Though this improvement is not evaluated in this work

4 Design

This section will describe the step-by-step design of the cloud architecture, hosting the system described above, on the Amazon Web Services infrastructure. An overall solution will be presented first with additional modifications and use cases discussed in the end of this chapter.

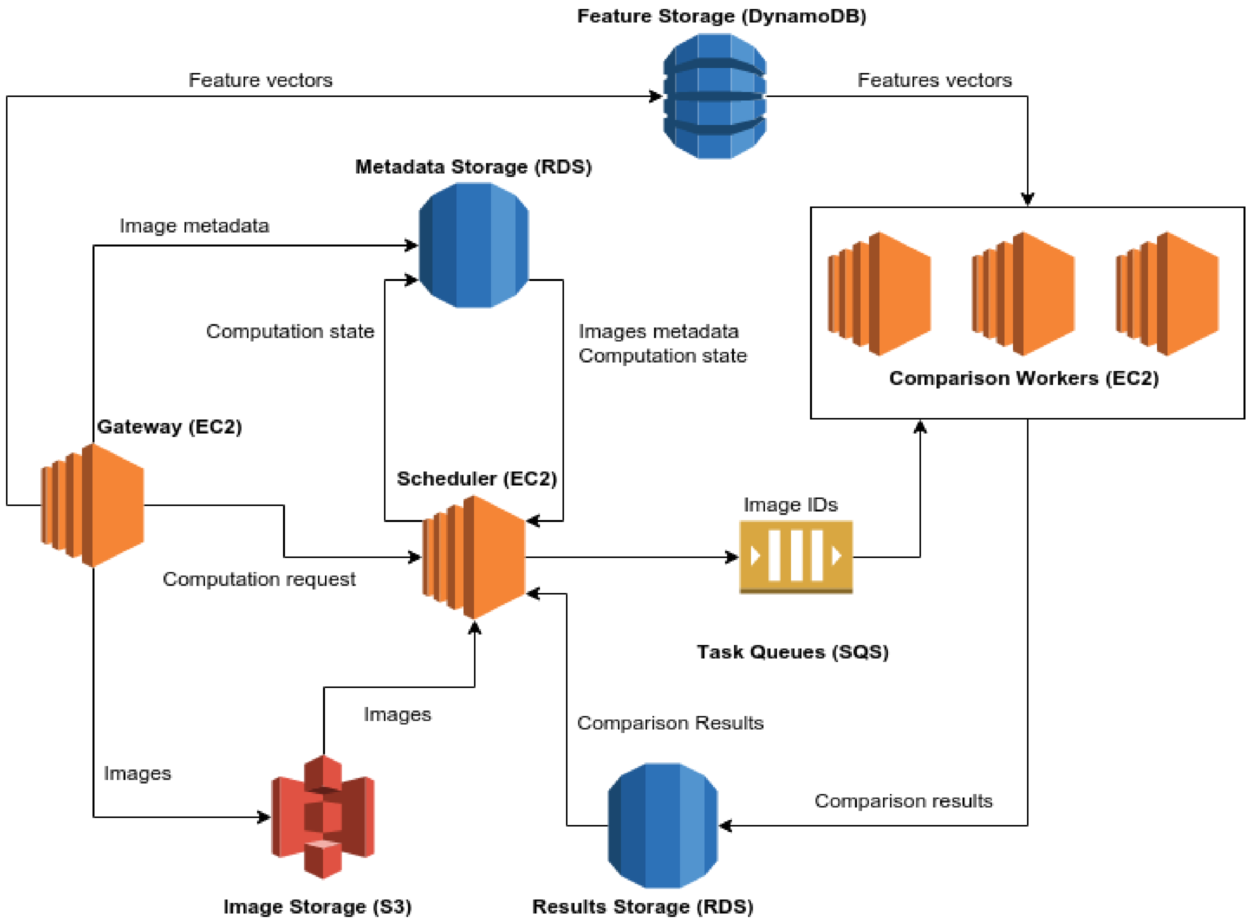

Figure 6: AWS architecture design

The high-level architecture design is depicted in the figure. The arrows symbolize the data flow in the system (data, objects, messages) with **Gateway** being the exposed system interface.

4.1 Gateway

Gateway is the only component exposing a public interface of the system. This service is responsible for processing all the user requests and triggering the computation tasks.

The following set of REST endpoints is exposed via this service:

- **/images** [POST] - Uploads the image to an **S3 bucket** (image storage) with tags and performs the image pre-processing tasks.
- **/images/[id]** [DELETE] - Deletes an image from the system.
- **/compare** [POST] - Schedules 1-N or N-N comparison. The POST request contains the comparison type and the image *id*, in case 1-N task was selected.
- **/results/[id]** [GET] - Gets the results of a computation with selected *task-id*[31].
- **/duration/[id]** [GET] - Gets the duration of the selected task[32].

Image pre-processing

The following operations are executed when uploading a new image to the system:

1. Upload the image to the **Image Storage**
2. Extract the feature vectors of the image and save them into **Feature Storage**
3. Save the image metadata to the **Metadata Storage**
4. Return the image id to the user

As described above, there are metadata saved for each image in the system. These contain the following information:

- id - unique identifier of the image
- image url - link to the S3 object
- tags - tags for the image (optional), specified by user in the POST request
- features - key to the extracted image features, one entry for each algorithm used in this extraction[33]

[31] When triggering a computation task, this task is assigned a unique ID.
[32] Used for benchmarking the system performance
[33] As mentioned in the beginning of this section, we can only compare features obtained by the same algorithm against each other

The scheduling endpoints just check the validity of the request (image existence), create a unique id for this task, which is saved to a database and sends a message to the scheduler which is responsible for these operations.

The services itself run on a single EC2 instance since it is not expected to handle large amount of image uploads at once and more than one comparison task running at the same time.

In case that large amount of images would need to be uploaded and processed at the same time, we could deploy this service to ELB or forward this pre-processing task to a dedicated set of workers connected to the Gateway by a queue.

4.2 Data Storage

S3 Standard was selected as an **Image Storage** solution since the data storage is not expensive[34] and there will not be many frequent operations over the saved images. The computation only uses the image features so the images themselves are only used for displaying the results.

As for the **Metadata Storage**, a MySQL database running on an RDS instance is used, since these data are relational and we need to query them by various parameters (algorithm choice, tags etc.). The *db.m4.large* instance type is going to be used for hosting this database. The comparison id with current status of the operation is kept on this instance as well.

DynamoDB was selected for the **Feature Storage**. The feature vectors are always queried by their hash-keys, which are stored in the Metadata Storage, so these Dynamo tables are only used as a key-value store.

Finally, the results of the computation are going to be persisted in a relational database as well **Results Storage**, since we need to be able to query them and sort them by multiple parameters. There could be potentially up to N^2 records in this table since each image comparison produces a unique score. Fortunately, we don't need to store all of them but only those that exceed some minimum threshold (we are only interested in images that are similar). Since the amount of records

[34]Compared to the other costs associated with the system's workflow. The exact pricing calculation will be presented at the end of this chapter

in the database is considerably larger than with the metadata Storage, a bigger instance type will be used to host this database: *db.m4.xlarge*

Let's consider a scenario with 10^6 images in the dataset. In addition to that, let's assume that there is a 0.01% chance that two images have a similarity[35]. This is the amount of data that will be saved in these storage services[36]:

Type	Entities	Item size (kB)	Size (GB)	Storage price
Images	10^6	500	500	\$15 monthly
Metadata	2×10^6	0.1	0.2	\$0.175 hourly (\$26 monthly)
Feature vectors	2×10^6	10	20	\$2.5 monthly
Results	10^8	0.1	10	\$0.350 hourly (\$52 monthly)

Table 26: Storage cost estimation

4.3 Scheduler

The main purpose of **Scheduler** is to distribute the workload between multiple workers and hence acts as a computation manager. Since the scheduling part is not scalable by nature, it runs on a single EC2 instance.

After receiving a computation request from the **Gateway**, Scheduler starts loading image metadata from the **Metadata Storage** and groups them together into messages. These messages are then sent to the **Task Queue** and picked up by individual workers. Each computation request (**1-N** or **N-N**) is assigned a unique ID and saved into a relational database (Metadata Storage instance is used for this purpose) together with the computation state (in progress, finished etc.).

Each message contains the following payload:

- Task id - id of the current computation task
- Multiple sets of image keys - keys pointing to the feature vectors. Each set contains only keys associated with one feature extraction algorithm (eg. ORB)

[35] Exceeding the predefined threshold

[36] All the item sizes shown in this table are estimated from sample data usage. These values should represent an average entity size.

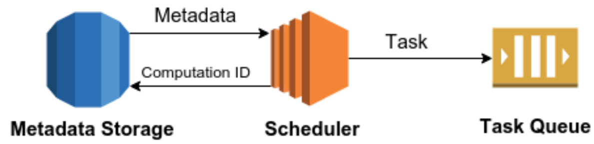

Figure 7: Scheduler data flow

There are important aspects of scheduling the worker tasks: the total number of images chosen for the comparison and the number of images in every task (message).

As for the first one, we probably don't want to compare all the images in the dataset against each other (in the N-N comparison task) since this would mean that billions of individual operations would have to be computed. Tags are used to reduce the total number of tasks. There is no point in comparing an image tagged as 'car' against another marked as 'building'. This could reduce the number of operations drastically since it is expected to have photographs from many areas of interest.

As for the number of images included in one message, we need to consider how the feature vectors will be loaded from the **Feature Storage**. Although it is possible to load them one by one, this approach would be highly inefficient. Instead, batch operations are going to be used over the feature DynamoDB table. Amazon allows *batchGet*, which returns up to 100 entities with the total size 16MB with a single request[37]. This means that we can put 100 images in a single message to reduce both the number of enqueued tasks and also minimize the traffic between workers and DynamoDB. The *batchGet* operation will have to be performed for every set of images corresponding to one feature algorithm.

The total number of DynamoDB operations can be computed using the following equation:

$$n_o = \frac{N \times n_c \times n_a}{n_t \times m}, \tag{1}$$

where N is the total number of images in the dataset, n_c the number of comparisons (eg. 1 or N), n_a the number of used algorithms (eg. 2), n_t the number of topics and m the size of one message (number of entities loaded in one batch operation).

The following table shows the number of messages and DynamoDB operations for **1-N** and **N-N**

[37] `http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_BatchGetItem.html`

comparison tasks. In the scenario with 10^6 images and two algorithms used for the feature detection, we assume that all pictures contain tags from 100 topics which are distributed uniformly (so each image will be compared against up to 1% other images in the dataset).

Comparison	Messages	DynamoDB batch operations
1-N	100	200
N-N	10^8	2×10^8

Table 27: Operation count estimation

4.3.1 DynamoDB Read Capacity

When estimating the cost of the DynamoDB operations and traffic we need to get back to the pricing model of this service, as described in the first part of this work. Each Dynamo table works with provisioned read and write throughput capacity which defines the number of reads and writes per second. One read capacity unit covers 4 kB of item size. With the average size of one stored feature vector being approximately 10 kB and retrieving 100 such vectors in one batch read, we need 500 read capacity units per operation. If we use eventually consistent reads[38] instead of consistent reads, we get double the effect of a single capacity unit and hence use only 250 capacity units per single batch operation.

Moreover, it is important to note, that read capacity defines the total number of reads per second but DynamoDB is charged by hour so we can spread the workload over longer time periods.

The following equations describe the price computation for reading feature vectors from the Feature storage:

$$r_o = \frac{m \times s_v}{8}, \tag{2}$$

where r_o is the number of reads in a single operation (batch read), m is number of entities in one batch (eg. 100) and s_v is the average size of one entity (eg. 20kB). The result is divided by the maximum size of one eventually consistent read (8kB).

Now, we need to compute the total number of read operations when performing one big task:

[38] `http://aws.amazon.com/dynamodb/faqs/`

$$n_r = c_o \times n_o, \tag{3}$$

where n_r is the total number of required reads, r_o the number of reads in one batch operation and n_o the total number of batch operations, as computed in the previous equation.

Finally, the price can be obtained with the following equation:

$$price = \frac{n_r}{3600} \times \frac{p_h}{50}, \tag{4}$$

where p_h is the hourly price for 50 read capacity units. It is important to note, that the capacity units are distributed uniformly across the one-hour time periodss and we therefore need to divide the result by the total number of seconds in one hour[39].

Comparison	**Batch operations**	**Reads**	**Cost**
1-N	200	50000	\$0.0018
N-N	2×10^8	$2 \times 250 \times 10^8$	\$1800

Table 28: DynamoDB operations cost estimation

The table above estimates the cost of read operations when performing one huge comparison task. The hourly read throughput will depend on the number of workers and the total time of the computation. The overall summary of these parameters will be given at the end of this chapter.

Moreover, when performing the N-N comparison task, the same image features are used in every 1-N iteration. It might be convenient to cache those features in local memory to avoid loading them repeatedly. Therefore, each **Worker** has a feature vector cache which temporarily stores a certain amount of features.

4.4 Workers

Workers perform the actual comparison operation between two images, respectively their feature vectors. It is connected to a task queue from which it consumes messages with keys of the image vectors to be compared.

[39] 1 read capacity units guaranties 1 read operation per second

After processing the message, it connects to the **Feature Storage** and downloads all the data needed to perform the task. Once finished with comparing the images, results exceeding a predefined threshold indicating a match are saved to the **Results Storage**.

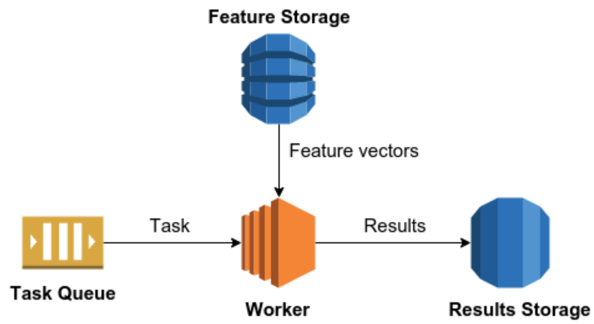

Figure 8: Worker data flow

In order to speed up the processing, each worker performs several tasks simultaneously in multiple threads. Another thread is dedicated solely to processing the task messages and loading the data from the database. This way, we can reduce the time when waiting for data to be processed.

Getting back to our example with 10^6 tagged images and 100 topics, this table shows the computation time estimate in case of running 10 instances (*c4.large*), each with 8 threads dedicated to the comparison operations. Moreover, the time of a single matching operation[40] is estimated to be 20ms.

The equation for computing the total number of time for one big comparison task looks as follows:

$$t_{total} = \frac{N \times n_c \times n_a}{n_{to}} \times \frac{t}{n_i \times n_{tr}}, \tag{5}$$

where N is the number of images in the dataset, n_c the number of comparison tasks (either 1 or N), n_a the number of algorithms used (eg. 2), n_{to} number of evenly distributed topics in the dataset, t the average time of one comparison operation (eg. 20ms), N_i the number of instances and n_{tr} the number of processing threads of each instance.

Comparison	**Operations**	**Total time**
1-N	20000	5 seconds
N-N	2×10^{10}	58 days

Table 29: Worker computation time estimation

[40] These benchmarks were obtained by testing the ORB feature detector in local environment similar to the instances described above

Note, that these estimates are rather optimistic since we are assuming that there are no idle waiting times and and each thread is 100% utilized at all times.

Finally, the price of running the worker instances can be computed as follows:

$$price = n_i \times t_h \times p_h, \tag{6}$$

where n_i is the number of instances, t_h the total number of hours of one comparison task rounded up and p_h the hourly price for running one instance (*c4.large*).

Comparison	**Total time (hours)**	**Cost**
1-N	1	$1.34
N-N	1392	$1865

Table 30: Worker computation cost estimation

The total price can be further reduced by using reserved instances, as described earlier in this work, by up to 30%.

4.5 Messaging

There are several possibilities to consider when selecting the messaging service in the proposed system. We could either use SQS, an Amazon's message-queue-as-a-service, or run a third party queue on an EC2 instance. Both of these options will be explored in this section.

4.5.1 SQS

Messages in SQS are strings up to 256kB in size and can be sent in bulks[41]. SQS guarantees at-least-once delivery. If a message is received, it is blocked for a specified period of time and unless deleted within that period, it will become available for delivery again. That means that no messages will ever be lost, since SQS is a replicated queue, but there is a potential risk of processing the same message twice.

Getting back to our example with 10^6 images in a dataset, we can easily compute the cost for sending the messages during one comparison task. In order to do so, we need to compute the size of one message bulk. Each message will contain 100 feature vector ids encoded in a string array.

[41] With the total limit of 256kB for the whole bulk

With ids having approximately 20 characters, we won't exceed 20kB in size of a single message. Each bulk will therefore have up to 200kB.

Comparison	Bulks	Size	Price
1-N	10	2MB	free
N-N	10^7	2000GB	$180

Table 31: SQS cost estimation

Since each worker polling the messages in parallel with performing the comparison computation and since each message contains 100 image references, it is safe to assume that there will not be any delays with workers waiting for new content to be delivered.

4.5.2 Hosting a Queue

Amazon SQS is a hosted service, so there is no need to handle any operational aspects like monitoring, administration or security. Moreover, SQS can scale almost unlimitedly.

However, there might be reasons to consider hosting your own messaging system - for example when fast response times and extremely low latencies are required. The most typical queues used for this purpose are RabbitMQ, ActiveMQ or ZeroMQ.

RabbitMQ implements a broker architecture so messages are queued on a central node. This makes the queue very easy to use and deploy but also leads to limited scalability. A cluster of RabbitMQ instances, can be configured, however.

ActiveMQ is very versatile messaging system which can be deployed both with broker or as a P2P topology. It offer wide range of functionality, including Message Groups or Virtual Destinations[42].

ZeroMQ is a light-weighted messaging system designed to provide low latencies and high throughput. Contrary to RabbitMQ, ZeroMQ operates completely without brokers. The downside is that advanced scenarios are slightly more difficult to implement since various pieces of the framework need to be combined to achieve the goal.

[42]This functionality is not relevant to the scenario described in this work, though.

Every messaging system described in this section supports replication and message persistence. These features needs to be configured manually, though. These queues would need to be hosted on its own EC2 instance (or several instances in case a whole cluster was needed).

One of the important aspects to note is the number of messages that can be queued at the same time. The number and size of the messages are limited by the parameters of the instance hosting the queue. **Scheduler** would potentially need to monitor the current state of the queue and delay sending additional messages when getting close to such threshold.

Moreover, unlike SQS which is managed and charged by message, the instance with a custom queue would be running the whole time, regardless on the actual workload.

These messaging systems will not be discussed into detail, since it is not in scope of this work. Number of publications have been written to cover this topic [15] [2].

The following table shows an estimated monthly cost when running a single EC2 instance with a queue[43]:

Instance type	**Monthly cost**
t2.medium	37
t2.large	74

Table 32: Messaging cost estimation, US region

4.6 Pricing Summary

The overall architecture design for the described system was presented in the previous section. Several choices were made in terms of services used for the individual components with their pros and cons.

This part presents the summary of all the costs associated with a full **N-N** comparison task discussed earlier. For this purpose, the same scenario as always will be considered: 10^6 images in a dataset with 100 uniformly distributed topics (tags), 10 instances using 8 computation threads and

[43]It does not matter which messaging system is used since it does not affect the cost of the instance

0.01% images sharing significant similarity[44].

As evaluated earlier, such computation would run approximately for two months and the following table shows the cost for all the system components:

Service	Cost
Worker instances	$1800
Image storage	$30
Image metadata storage	$52
Feature vectors storage	$5
Results storage	$104
Provisioned throughput capacity	$1800
Messaging	$180
Total	$3971

Table 33: Cost estimation summary, US region

It is important to note, that these costs could be reduced by using reserved instances and reserved capacity by $20 - 30\%$[45].

4.7 Special Use-cases

This section will describe several special cases which may occur during the usage of the system and discuss possible solutions to them in terms of the proposed architecture design.

4.7.1 Adding New Images During Task Computation

The computation of one **1-N** or **N-N** task might take large amount of time, ranging from minutes to weeks, as described earlier in this section. This means that dynamic addition of new images during a computation in progress has to be handled as well.

Fortunately, each computation task is assigned a unique ID, which is saved to a database together with the current state. If a new image is uploaded to a system via **Gateway**, the **Metadata storage** has to be first queried for any computations in progress. If there are any, the **Gateway** sends a computation request containing the computation ID to the scheduler so the new image can be added to the running task.

[44]The scenario parameters significantly affects the architecture design and cost parameters of the system. It is not in scope of this work to evaluate all the possible configurations so this candidate was selected to represent a 'reasonable' input data.

[45]`https://aws.amazon.com/ec2/purchasing-options/reserved-instances/`

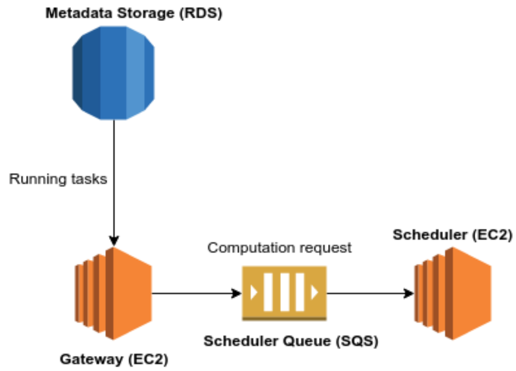

Figure 9: Dynamic image addition

4.7.2 Short Computation Over a Dataset Without Tags

In the previous sections, we proposed a cloud architecture of an application processing large datasets of tagged images. When estimating the time and cost of running the system, we made several assumptions:

- The images are all tagged
- There is a wide range of uniformly distributed tags which determine how are images compared against each other
- The computation itself can run for several weeks for the **N-N** comparison task

However, these assumptions might not be realistic in some scenarios. First of all, if there are no tags in the collection, all the images in the dataset will have to be compared against each other[46] which would drastically increase the computation time even with large number of instances. Secondly, it might not be feasible to wait for the computation for such a long time.

Let us image the following scenario: **N-N** comparison is triggered on regular basis on smaller image sets without tags. The computation has to be finished under 12 hours since the results have to be prepared for the next day. How many images can be in one dataset to fulfil these constraints?

The answer to this question depends mainly on the number of instances processing the workload. The maximum number of images in the dataset, when performing the **N-N**comparison task, can be

[46] Previously, we only worked with small subsets of the images because only pictures with same tags were compared

obtained by the following equation[47]

$$N_{max} = \sqrt{\frac{T \times n_i \times n_t}{t}}, \tag{7}$$

where T is the available time for the task (e.g. 12 hours), n_i the number of running instances using n_t threads each and t the time for performing a single comparison operation (e.g. 20ms).

The following table illustrates the maximum dataset size depending on the number of instances processing the workload. The processing time of a single comparison operation is, similarly to the previous section, estimated to be 20ms and every instance uses 8 threads dedicated for the computation. The total time reserved for the task is 12 hours and the instances used for the computation are *c4.large*[48].

Instances	**Dataset size**	**Cost**
10	13145	\$10.34
100	41569	\$103.44
1000	131453	\$1034.4

Table 34: Maximum dataset size estimation

4.7.3 Populating the System Database

So far, all the problems and corresponding solutions discussed in this chapter worked with the assumption that the application is set-up and there is a dataset of processed images already stored in the system. There might be situations, however, when it is required to populate the database with an existing set of images before launching the computation. Although it is possible to upload the images one by one using **Gateway**, this process is not automated.

One way to process such workload would be converting **Gateway** into a worker instance with horizontal scaling capabilities and creating new component for the coordination of the dataset processing.

The following diagram shows a simple architecture for this solution:

[47] The overhead created by loading data from the database is neglected here since these operations will run in parallel with the computation as described in the previous section. This will, however, still give only the optimistic upper bound of the result.

[48] On-demand prices are used for the cost estimation.

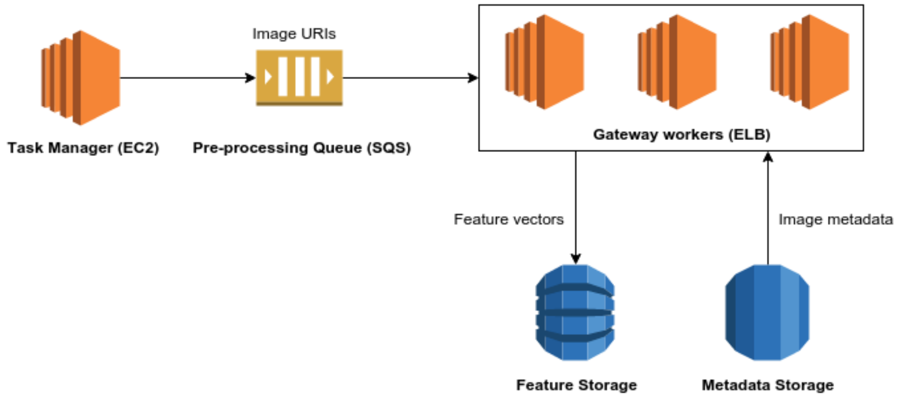

Figure 10: Populating the system database

Task Manager, hosted on an EC2 instance, is exposing the REST interface. It accepts an URL pointing to an image repository. This repository is scanned and all the image URIs are sent to the **Gateway workers** via a **Pre-processing Queue**. The image pre-processing itself is than identical to the one performed by the **Gateway** service discussed in the previous section.

5 Implementation

In this section, a simplified implementation of the system described in the previous part will be discussed with focus on the technology stack used and challenges faced during the development. Benchmarks and summary of the system performance will be given in the final part of this work.

It is not the aim of this work to discuss the structure of the individual components in the system, since these are mostly micro-services and the overall architecture has already been described in the previous chapter[49].

5.1 Technology Stack

The whole system is implemented in Java, since it has a strong support in AWS SDK and offers a seamless integration with all the required AWS services. The components (Gateway, Scheduler and Workers) are written in Spring Boot[50] framework, that enables simplified Spring based application design and deployment.

Several Spring modules are used in most of the system components:

- **Spring JPA** providing easy access and manipulation with entities persisted in a relational database
- **Spring Web** for exposing REST endpoints
- **Spring Actuator** including a number of additional features for monitoring and management of the application

All components are packaged as WAR[51] files and deployed to Elastic Beanstalk Tomcat instance[52]. Gateway and Scheduler run inside a *Web application environment*, enabling HTTP(S) communication and Worker uses *Worker environment* with access to a predefined SQS Queue[53]. Furthermore, Workers are deployed in a load-balancing mode so additional instances are launched based on the workload (either an instance utilization or number of messages in the queue).

[49] The source code of the discussed implementation is available as an attachment to this work
[50] `http://projects.spring.io/spring-boot/`
[51] Using Maven: `https://maven.apache.org/`
[52] `http://tomcat.apache.org/`
[53] `http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html`

5.2 Packaging OpenCV

One of the main problems, when deploying the system to the cloud, was using OpenCV on the remote instance. OpenCV supports desktop Java development since version 2.4.4 so it can be compiled into a JAR file and added to the build path of the application. However, there are several problems with this approach:

- OpenCV has to be installed on the machine
- Path to the native OpenCV libraries needs to be provided in the Java environment, these may vary based on the operation system
- OpenCV JAR has to be compiled based on the native libraries version and added to build path manually since there is no official Maven artifact

This complicates the deployment since a script for installing, compiling and linking OpenCV to the ELB Tomcat environment.

Fortunately, there is a third party solution - OpenCV Java bindings packaged with native libraries provided as a Maven artifact[54]. Although this project is no longer supported[55], it provides all the necessary capabilities for demonstrating the system functionality.

5.3 Serialization of OpenCV Objects

In order to perform the computation over large image dataset efficiently, only the extracted feature vectors are used instead of the image files. That means that these features needs to be persisted in a database. Typically, Java objects can be stored as a sequence of bytes using process called *serialization* [20].

Feature vectors are stored in OpenCV MAT data types[56]. Unfortunately, they are not implemented as *serializable* objects so this functionality has to be added manually. All the extracted feature vectors, represented as MATs, are therefore converted into JSON strings and saved in a database in this format. Once loaded, these JSON objects are *deserialized* into OpenCV MAT representation again.

[54] `https://github.com/PatternConsulting/opencv`
[55] It supports OpenCV 2.4.9
[56] `http://docs.opencv.org/java/3.0.0/org/opencv/core/Mat.html`

5.4 Selecting Feature Extractor

There are number of algorithms that could be used for the feature detection and descriptor extraction, as described in the previous sections. Unfortunately, the actual choice is limited by the fact that SURF and SIFT algorithms were removed from the official OpenCV distribution and became available only as part of a non-free module due to their licensing.

From the remaining algorithms, ORB and AKAZE are available for this task. ORB was chosen as a reference for the system benchmarking, although both algorithms offer satisfactory performance and results[57]. As discussed in the design section, there could be several feature detectors used in the system but only one was chosen for this implementation to avoid unnecessary complexity (and costs associated with running the application).

5.5 Possible Adjustments

There are several possibilities how to further improve the system performance. These will not be implemented and tested as part of this work, although some of them will be mentioned for future consideration.

5.5.1 Dedicated Workers

One of the key drawbacks of this implementation is that the workload is distributed randomly between all the workers. This decreases the effect of caching feature vectors on the worker instances since the cache is not accessed so frequently.

Instead of having universal workers fetching messages from one task queue, there could be several queues dedicated to different tags or groups of images and corresponding groups of dedicated workers. This would improve the resource utilization, especially with non-uniformly distributed tags because each group of workers could contain different number of instances.

5.5.2 Elastic MapReduce

Another possibility is to use a completely different approach to the problem. Instead of designing our own application architecture, a MapReduce [10] service from Amazon called *Elastic MapReduce*

[57] `http://docs.opencv.org/3.0-rc1/dc/d16/tutorial_akaze_tracking.html`

(EMR) could be used. In this way, a managed Hadoop framework with cluster of EC2 instances is provided by AWS.

However, since MapReduce is usually used for processing large amounts of text data at once, it might not be a suitable option for the **N-N** comparison task since multiple jobs would need to be executed in a sequence. The only alternative would be providing an input consisting of task messages and letting the cluster instances load the data from DynamoDB. This solution is very similar to the proposed architecture, though.

5.5.3 Image Comparison Adjustments

The proposed architecture is capable of comparing images against each by using their extracted feature vectors (image descriptors), extracted using one or more algorithms described in the previous sections. However, as mentioned earlier, this method does not guarantee high accuracy and there is a risk of many false positives and false negatives when using this method independently.

One of the possibilities to improve the comparison accuracy is to add another metric during computing the score of a match. Various methods based on histograms, for example, could provide this additional metric [21]. These would need to be precomputed during the image pre-processing phase and store in a persistent storage (like DynamoDB) to enable fast access during the computation. This would, however, increase the traffic[58] between workers and the storage services since more data would be needed to perform the comparison task.

This approach, combined with using multiple feature extracting algorithms, could be used for smaller datasets or **1-N** comparison tasks where high accuracy is required, though.

[58] not mentioning the time performance and costs

6 Testing

The implementation of the proposed system was tested against a prepared dataset to verify the expected system behaviour. It is, however, important to note, that the costs associated with running the system are a limiting factor so the size of the dataset and number of instances running during the computation were adjusted to meet this constraint.

During the benchmarking, the following set-up was used to test the system:

- 2 workers running on *t2.micro* instances with 5 working threads each
- RDS database instance running on *t2.micro*
- messages containing 10 comparison tasks each[59]
- one algorithm used for the feature matching (ORB)
- dataset containing two topics with the same number of images

The expected and actual time performance of the system will be presented in this section for the **N-N** comparison. The same equations as before are going to be used for the estimation:

$$t_{total} = \frac{N \times N \times n_a}{n_{to}} \times \frac{t}{n_i \times n_{tr}}, \tag{8}$$

where N is the number of images in the dataset, n_a the number of algorithms used, n_{to} number of evenly distributed topics in the dataset, t the average time of one comparison operation, N_i the number of instances and n_{tr} the number of processing threads of each instance .

The average time needed for one feature comparison operation, measured on the running *t2.micro* instances, is 30ms. This is slightly more than presented in the previous section, which is caused by the fact that the initial estimation was done on a more powerful machine[60] and it also includes the time need to load the feature vectors from DynamoDB (approximately 5ms per entity). A dedicated thread for loading the vectors was not used in this scenario in order to increase the utilization of

[59] The proposed number of tasks in each message is 100, as described in the previous section, but it makes sense to lower this limit for smaller number of workers in order to distribute the work more fluently.

[60] In a scenario with very large datasets, as presented earlier, *c4.large* instances with better performance would be used

all the available threads.

The following table shows the results for multiple dataset sizes:

Dataset size	Operations	Expected time	Actual time
300	45000	135s	138s
500	125000	375s	391s
800	320000	960s	1012s
1000	500000	1500s	1631s

Table 35: System benchmarks

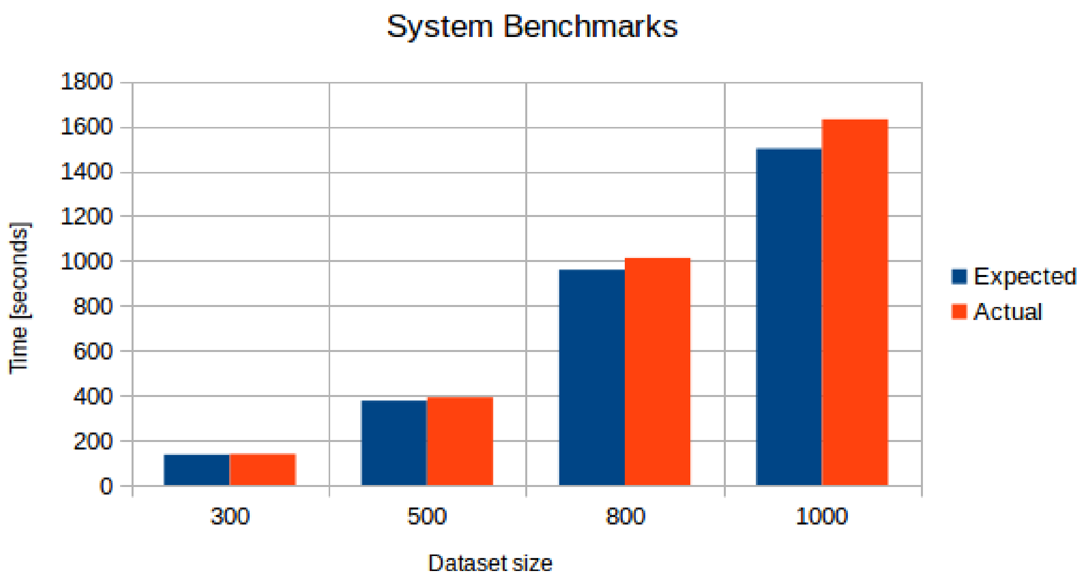

Figure 11: System benchmarks

Note, that *operation*, in this sense, it the comparison between two sets of feature vectors representing the images. These typically consist of 500 vectors being compared against each other so the actual number of arithmetical transactions is much higher.

The results show that the reference implementation meets the proposed criteria in terms of time-performance. The actual time needed for the computation exceeds the estimated value by only a few percent. However, it is important to note that this is a very small example, compared to the scenario with 10^6, and that caching can affect the computation time significantly. Nevertheless, the implementation results show that the application can be deployed and operate with performance close the expected constraints.

As for the accuracy of the image comparison, by looking at the best 20 results, there were at most 1-2 false positives after each comparison task.

The following table shows the 5 best results of the comparison task with 1000 images in the dataset:

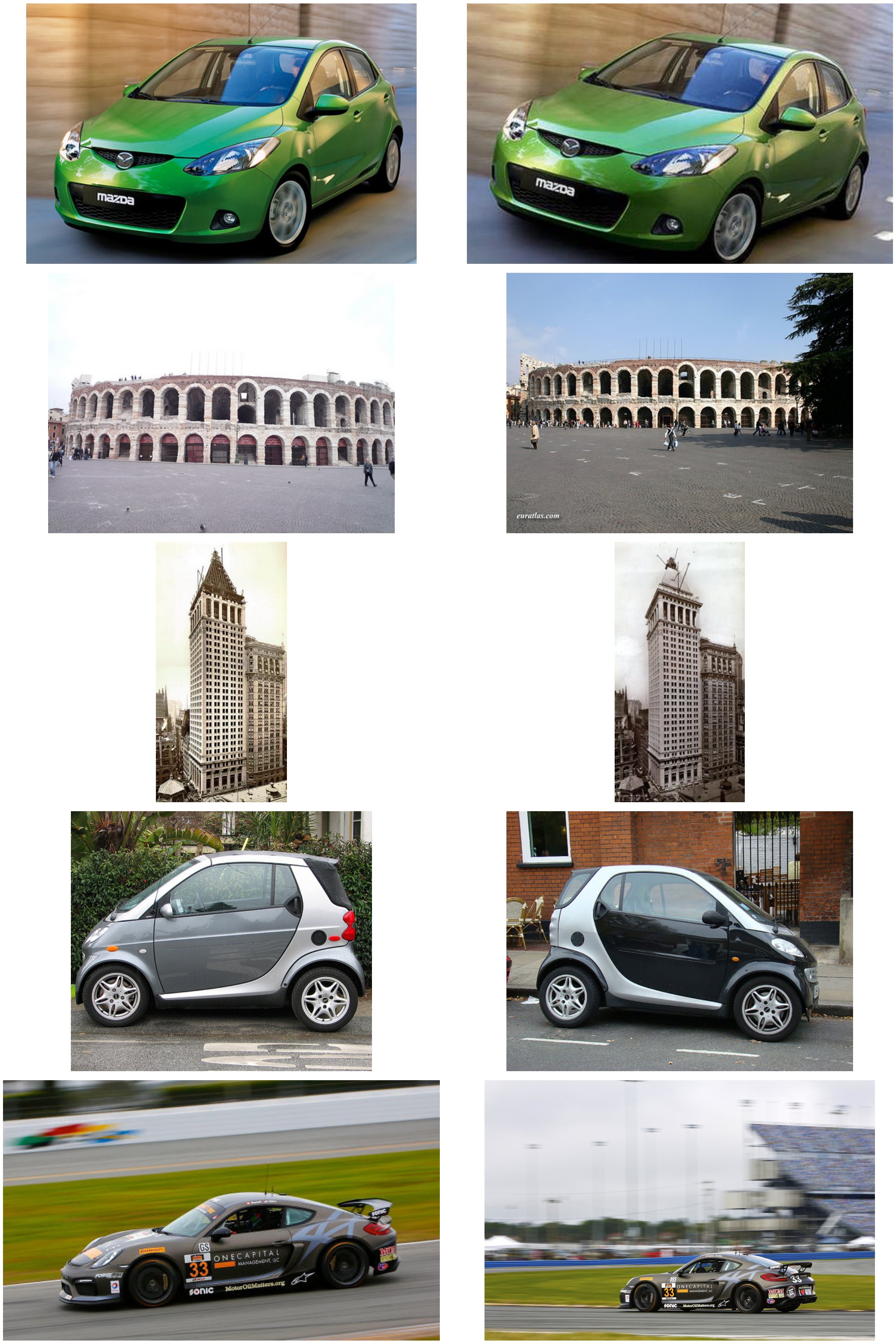

Table 37: Top 5 results

7 Summary and Conclusions

The aim of this work was to give an introduction to cloud computing and to show how a specific computationally intensive application could be developed using the state of the art technologies.

After a brief overview of the cloud computing problematic, several vendors of cloud technologies, namely Amazon, Microsoft, Google and IBM, were selected and compared in terms of the most common services and their corresponding pricing models. Pricing is an important aspect when designing cloud solutions, since cloud applications usually work with huge amount of data (or traffic, availability etc.). The operational costs could differ greatly based on the chosen technology stack.

In addition to that, the discussed vendors sometimes offer discounts to their services. These discounts have been reviewed and considered in the comparison section as well.

All aspects of the application design in the cloud were considered, namely compute capacity, object storage, SQL and NoSQL database solutions and messaging.

In the second part of this work, a specific problem, searching for similarities in large datasets of images using the OpenCV library, is introduced and a model scenario for this task is considered. A cloud architecture for computing this task is shown, using the Amazon AWS technology stack. Each problem domain is explored in terms of costs, time performance and optimal utilization of resources. This includes scalability, bottlenecks and other limitation factors faced during cloud development. All the system parameters are estimated using a model scenario with million images divided into several uniformly distributed sets of images defined by tags.

In addition to that, a modified version of the problem, with only a limited time available for each computation, was explored and a theoretical setup and dataset size was computed to meet the defined goals.

A simplified implementation of the proposed architecture is shown in the following part. The aim of this section was to discuss several interesting problems faced during the development rather than focus on detailed description of the code[61]. The application was deployed to AWS as a fully functional system.

Finally, several tests with predefined system set-up and various sizes of the input image dataset were performed and the obtained benchmarks were compared to the expected values, based on the design section of this thesis. These benchmarks showed that the system behaviour met the predicted results and thus verified the proposed architecture for given problem.

7.1 Future Work

The technologies in the cloud computing area are constantly evolving and new solutions are being introduced almost on daily basis. It is thus important to keep track of the up-to-date services and adapt the applications to given problems.

In the previous section, several possibilities for adjustments of the application architecture were presented to increase the efficiency and resource utilization. These could be implemented to further improve the performance and decrease the cost of running the system with large workloads.

[61] The source code can be found as an attachment to this work.

References

[1] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. Deconstructing amazon ec2 spot instance pricing. *ACM Transactions on Economics and Computation*, 2013.

[2] Faruk Akgul. *ZeroMQ*. Packt Publishing Ltd, 2013.

[3] Pablo Fernandez Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *Computer Vision–ECCV 2012*, pages 214–227. Springer, 2012.

[4] Pablo Fernandez Alcantarilla and TrueVision Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[6] Trieu C Chieu, Ajay Mohindra, Alexei A Karve, and Alla Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*, pages 281–286. IEEE, 2009.

[7] Jiang Dejun, Guillaume Pierre, and Chi-Hung Chi. Ec2 performance analysis for resource provisioning of service-oriented applications. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, pages 197–207. Springer, 2010.

[8] Thomas Deselaers, Daniel Keysers, and Hermann Ney. Features for image retrieval: an experimental comparison. *Information Retrieval*, 11(2):77–107, 2008.

[9] Tanmay Deshpande. *Mastering DynamoDB*. Packt Publishing Ltd, 2014.

[10] Zacharia Fadika and Madhusudhan Govindaraju. Delma: Dynamically elastic mapreduce framework for cpu-intensive applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 454–463. IEEE, 2011.

[11] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010.

[12] Roger Jennings. *Cloud Computing with the Windows Azure Platform.* John Wiley & Sons, 2010.

[13] Luo Juan and Oubong Gwun. A comparison of sift, pca-sift and surf. *International Journal of Image Processing (IJIP)*, 3(4):143–152, 2009.

[14] Ben Kepes. Understanding the cloud computing stack: Saas, paas, iaas. *Diversity Limited*, pages 1–17, 2011.

[15] L Magnoni. Modern messaging for distributed sytems. In *Journal of Physics: Conference Series*, volume 608, page 012038. IOP Publishing, 2015.

[16] Maurício Marengoni and Denise Stringhini. High level computer vision using opencv. In *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2011 24th SIBGRAPI Conference on*, pages 11–24. IEEE, 2011.

[17] ABM Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*, 2013.

[18] Rafael Moreno-Vozmediano, Rubén S Montero, and Ignacio M Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, (12):65–72, 2012.

[19] Mohsin Nazir, Prashant Tiwari, Shakti Dhar Tiwari, and Raj Gaurav Mishra. Cloud computing: An overview. *Book Chapter of Cloud Computing: Reviews, Surveys, Tools, Techniques and Applications-An Open-Access eBook published by HCTL Open*, 2015.

[20] Lukasz Opyrchal and Atul Prakash. Efficient object serialization in java. In *Electronic Commerce and Web-based Applications/Middleware, 1999. Proceedings. 19th IEEE International Conference on Distributed Computing Systems Workshops on*, pages 96–101. IEEE, 1999.

[21] Greg Pass and Ramin Zabih. Histogram refinement for content-based image retrieval. In *Applications of Computer Vision, 1996. WACV96., Proceedings 3rd IEEE Workshop on*, pages 96–102. IEEE, 1996.

[22] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[23] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[24] Jurg Vliet, Flavia Paganelli, Steven Van Wel, and Dara Dowd. *Elastic Beanstalk.* " O'Reilly Media, Inc.", 2011.

[25] Uchit Vyas and Prabhakaran Kuppusamy. *DynamoDB Applied Design Patterns.* Packt Publishing Ltd, 2014.

[26] Yuping Xing and Yongzhao Zhan. Virtualization and cloud computing. In *Future Wireless Networks and Information Systems*, pages 305–312. Springer, 2012.

[27] Yulin Yao. Cloud computing: A practical overview between year 2009 and year 2015. *International Journal of Organizational and Collective Intelligence (IJOCI)*, 5(3):32–43, 2015.

[28] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

List of Figures

Content of CD

- **thesis.pdf** - full text of this work
- **src** - directory with source code of the application, contains installation notes inside README.txt