Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Evaluating the Profitability of the MediaWiki Application under different Cloud Distribution Scenarios

María Elena Alonso Mencía

Course of Study: Computer Science

Examiner: Prof. Dr. Dr. h. c. Frank Leymann
Supervisor: Dipl.-Inf. Santiago Gómez Sáez
Commenced: April 7, 2016
Completed: September 29, 2016

CR-Classification: C.2.4, C.4, G.1.2

Abstract

Cloud computing has gained popularity over the last years, causing a significant increase of available cloud offerings among providers. Therefore, this wide spectrum of options has led to an increment of possibilities for distributing applications in the cloud, by means of selecting specialized services to host each application component. Nevertheless, it also implies the need of finding the optimal solution depending on its purpose, usually based on future economical profitability. Nowadays, instead of considering an application as a whole when deploying it in the cloud, e.g. deploying whole application stack in a virtual machine, investigations focus on how to distribute the application components in heterogeneous cloud environments. Consequently, users have an even higher range of options and should carefully choose good decision criterion, going further than only considering the direct cost for the needed cloud instances. Some challenges are deriving a revenue model - as they tend to be application specific - and customizing the evaluation of different migration configurations of a real application with authentic data metrics. In this sense, this document uses utility analysis as it includes a non-directly countable element, preferences, and allows basing the decision on a trade-off taking into account other aspects which have an influence on the final performance such as users satisfaction or cloud instance availability under different deployment topologies. Therefore, the evaluation and comparison of different selected cloud offerings is possible and helps throughout the decision. This thesis presents an overview of state-of-the-art revenue models used nowadays on web applications and afterwards specifies the study and aims to apply the utility concept to evaluate a current application, MediaWiki, based on real data. Results show that this approach is more complex and differs from the one considering only the monetary expenses, pursuing a better balance between the possible business-technology conflict.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Cloud computing has become one important technology nowadays, offering multiple options for users. It has also advantages in small and large areas, easing the deployment of applications - or some of its individual components -, reducing initial inversions (e.g. physical infrastructure) and increasing scalability possibilities. Many service models have appeared recently and the technological scenery follows the *Everything-as-a-Service* model (*aaS), as [SAL16] refers to it. It offers flexibility to cloud users when deciding to migrate an application but, on the other hand, finding the optimal deployment is more complex.

Moreover, considering the different individual parts of the application highly increases the possible deployment combinations and widely extends the possible viable topologies. GENTL, TOSCA standard or Blueprints can be used to describe an application and its topology, depicting for example its components, interactions and inheritances. [ABLS13] distinguishes four possible migration types from partially making use of the cloud (types I - *replace component with cloud offerings* - and II - *migrate some of the application functionality to the cloud*) to completely migrating the application, where Andrikopoulos et al. distinguish two options: type III - classic migration, *migrating the whole software stack*, considering the application as one integral piece - and type IV - *cloudify the application* as a combination of different cloud services, with its associated complexity but also flexibility.

Therefore, as there exist multiple paths, it is essential to choose the optimal deployment decision. This can be made based only on total business expenses of all the needed cloud instances (according to the pricing models of the provider) but other more complex approaches are suggested. However, utility functions can have application on this issue, allowing any modelling specified and defined by the user depending on his preferences.

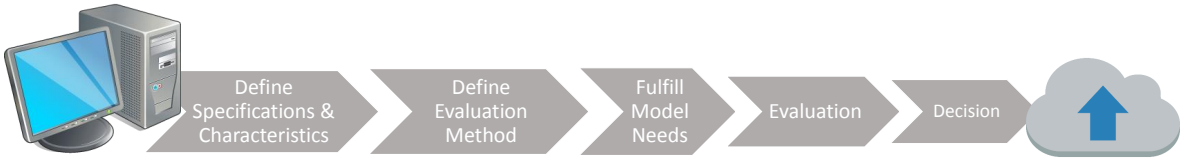

Figure 1.1: Migration of a Web Application to the Cloud Process Scheme

1.2 Problem Definition and Challenges

Utility-based analysis offers another step on the procedure addressed to optimize an application migration to the cloud, by considering other elements together with the cloud instance cost and allowing the evaluation of different metrics into one axis. User satisfaction, availability and adaptation costs, for example, can have an influence on final profitability and should be taken into account if a deeper approach is needed. However, infinite utility-based analysis may be applied, depending on the type of application and the factors that users focus on maximizing.

In the scope of this thesis, utility is used to analyze the trade-off between expected benefits - and its influences - and economical expenses. Most web applications aim to be profitable and generate some revenue through its use and, therefore, its business model should be studied. However, with new technological options and a general increase of connectivity at our society, new models have appeared and the first challenge is presenting some of the state-of-the-art and commonly used ones as the base for the analysis. There exist some utility approaches, for example related to economical decisions such as investments [Tho03] or more technological issues like provisioning storage systems [STFG08]. This thesis, as [Fre16], follows the path of applying this theory to cloud migration of applications but there is a need of studying it at a real case of web application which, intrinsecally to the evaluation process, also involves the challenge of collecting and analysing real data. MediaWiki has been chosen as the application under study on this thesis, following studies such as [SAL16] or [SALS15] which also used this example at their evaluation. By summarizing it, presented challenges are:

- Examine revenue models for web applications.
- Define a utility-based analysis suitable for cloud applications distribution.
- Evaluate a real case, after collecting and selecting real data and studying its influence on revenue.

1.3 Outline

This document is organized in 8 sections:

1. **Introduction**: Definition of the problem and objectives for the thesis. Listing of the thesis structure and some acronyms used in it.
2. **Fundamentals**: Necessary concepts and study areas for the thesis are clarified.
3. **Related Works**: Existing works related to the topic of the thesis are covered and compared with this document.
4. **Business Application Revenue Models**: Presentation of some state-of-the art revenue models with examples and comparison between their uses at different web application types.

5. **Wikipedia Revenue Analysis**: Examination of Wikimedia Foundation revenue model and MediaWiki topology. It also presents the used data collection and its evaluation. Derivation of monthly revenue models for the first semester of the year.
6. **Utility Functions**: Utility analysis designed for the evaluation of cloud computing applications at this thesis is introduced and defined.
7. **Case Study: MediaWiki and English Wikipedia**: Previous implementation is used to evaluate MediaWiki performance by using the collected real data from English Wikipedia and the utility analysis previously defined.
8. **Outcome and Future Work**: Obtained results are discussed and possible forwarding research areas are stated.

1.4 Abbreviations and Acronyms

This section lists some abbreviations and acronyms that are used throughout this thesis and may be useful for the reader:

API	Application Programming Interface
AWS	Amazon Web Services
CaaS	Communication as a Service
CMS	Customer Management System
CPC	Cost per Click
CPI	Cost per Impression
CPM	Cost per Mile
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSAR	Cloud Service Archive
DaaS	Data as a Service
DBaaS	Database as a Service
ERP	Enterprise Resource Planning
FAQ	Frequently Asked Questions
GCE	Google Compute Engine
GENTL	Generalized Topology Language
GUI	Graphical User Interface

HaaS	Hardware as a Service
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IBM	International Business Machines Corp.
ID	Identifier
IoT	Internet of Things
IT	Information Technology
JDBC	Java Database Connector
JRE	Java Runtime Environment
M	Million (i.e. 10^6)
MATLAB	Matrix Laboratory
NIST	National Institute of Standards and Technology
NumPy	Numeric Python
PaaS	Platform as a Service
PSO	Particle Swarm Optimization
QoS	Quality of Service
RDS	Relational Database Service
RUBiS	Rice University Bidding System
SaaS	Software as a Service
SciPy	Scientific Python
SLA	Service-Level Agreement
SQL	Structured Query Language
TCP-W	Transmission Control Protocol Web Benchmark
TOSCA	Topology and Orchestration Specification for Cloud Applications
URI	Uniform Resource Identifier Specification
VM	Virtual Machine
Wikim.	Wikimedia
XML	Extensible Markup Language

2 Fundamentals

2.1 Cloud Computing

The concept appeared for the first time in 2007 [NDDSD13] and since then, its definition, characteristics and possible applications have been widely studied. Many definitions of this emerging technology have been developed in the academic or business areas:

Buyya et al. [BYV+09] defined it as *a parallel and distributed computing system consisting of a collection of inter-connected and virtualised computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements (SLA) established through negotiation between the service provider and consumers*.

Madhavaiah et al. [MB12] stated that *cloud computing is an information technology-based business model, provided as a service over the Internet, where both hardware and software computing services are delivered on-demand to customers in a self-service fashion, independent of device and location within high levels of quality, in a dynamically scalable, rapidly provisioned, shared and virtualized way and with minimal service provider interaction*.

The National Institute of Standards and Technology [BGPCV12] sees it as *a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*. They distinguish among 3 service and 4 deployment models that are explained afterwards, and 5 essential characteristics:

- **On-demand self-service**: Resources required by the consumer are provided as demanded without the necessity of human interaction with the service provider.
- **Broad network access**: Standard mechanisms over networks are used to access the services available and, therefore, allows it through different devices (e.g. mobile phone, computer).
- **Resource pooling**: Resources are shared by multiple users in a dynamic way through a multi-tenant model but without any (or only at high-level) control of the location of the resources assigned.
- **Rapid elasticity**: Automatically resource scaling in and out depending on the increase or decrease in demand, done in a transparent way so that the consumer does not appreciate limits on the amount of possible capabilities.
- **Measured service**: Resources are controlled and optimized conditioned on measurements that depend on the type of computing capability used. Moreover, the usage of the resource can also be monitored and, consequently, adding transparency to service providers and customers.

2.1.1 Actors

Three main roles are distinguished in [VRMCL08] related to the deployment of cloud systems: The needed infrastructure is offered by the infrastructure providers to the service providers as a service, allowing them to transfer computing capabilities, with the associated economical and versatile advantages. This is the basic structure that allows the second group to offer through Internet their own software services in the cloud to the service users, closing the cycle.

On the other hand, [LBRK10] studies the actors of cloud computing from a business perspective and adds three new roles to the before mentioned actors (named *infrastructure providers*, *service providers* and *customers*):
Aggregate service providers (or aggregators) are at the same time customers and providers as they combine already-existing total or partial services creating new ones that are offered to users. If they aggregate data instead of services, guaranteeing it could be used by various cloud services, they receive the name of data integrators.
Platform providers offer the foundations for the deployment of applications as they serve as a service register offered by distinct providers.
Consulting is in charge of identifying services that are suitable to add value to the customer's business model.

2.1.2 Service Models

There are mainly three types of service models depending on the service requirements of the users [BGPCV12]:

- ***Infrastructure as a Service***(IaaS): Offers to the network architects storage, network, and processing tools, as the base for installation, development and running of software. It relies on virtualization technologies (being able to have *ad-hoc* systems [VRMCL08]) although the main disadvantage is the insufficiency of constant performance guarantees [LBRK10]. In this model, consumers can control the operating system and storage.

- ***Platform as a Service***(PaaS): Offers the means to application developers in order to deploy cloud applications using tools supported by providers (e.g. programming languages). The main disadvantage is the possibility of lock-in effects, causing dependency between the application developer and the used platform provider. In this model, consumers cannot control the lower layers of the system such as operating system and storage, but they can shape the applications.

- ***Software as a Service***(SaaS): Offers to the end users applications that rely on a cloud service. This software can be accessed over internet through a browser or a program interface. It could even be an aggregation of different cloud services but the end user will see it as a single one [LBRK10]. In this model, consumers cannot control the lower layers of the system such as operating system and storage, nor the applications.

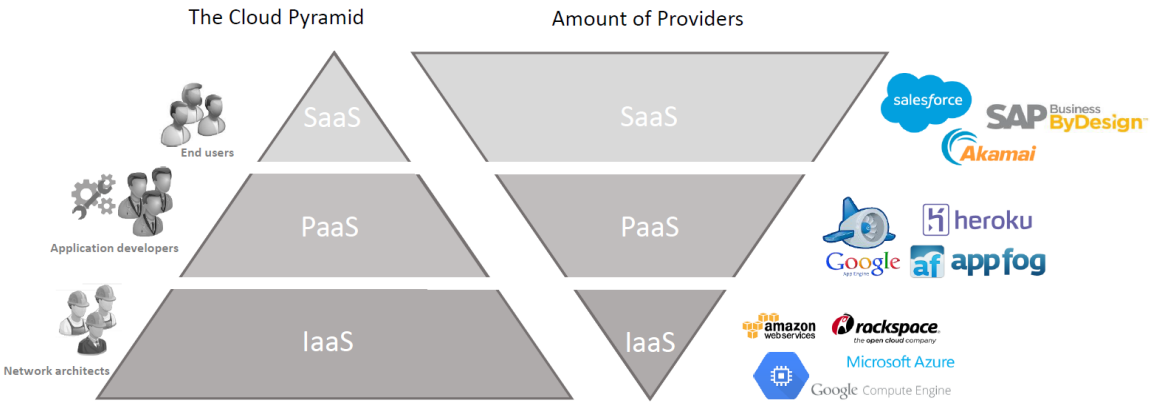

Figure 2.1: Cloud Computing

Figure 2.1 shows how these mentioned cloud service models are built one over another, closer to end users as going upwards in the pyramid. It is also shown that the amount of cloud providers increases as approaching closer-to-user solutions.

However, they are not the unique types as many other have been also studied: For example, *Data as a Service* (DaaS) and *Hardware as a Service* (HaaS) [WTK+08], *Communication as a Service* (CaaS) [LBRK10], *Database as a Service* (DBaaS) [LS10] or *Security as a Service* [VT14].

2.1.3 Deployment Models

Depending on the user requirements and preferences, different deployment models for the cloud system can be chosen [BGPCV12]. Each cloud infrastructure has specific characteristics in the access to the resources. The four deployment models presented are:

- **Public cloud**: Can be used openly by the general public. It is the most-used model and the provider has full control over aspects such as policy, value and charging model [DWC10]. Its management, ownership and operation is done by a combination of business, government or academic organization [BGPCV12].

- **Private cloud**: Can only be used by a single organization although its ownership, management or operation is not necessary done by it. The location of the infrastructure could be premise or off premise [DWC10]. Optimizing the already existing resources inside the organization and security worries such as data privacy are two reasons to choose this model [DWC10].

- **Community cloud**: Can only be accessed by a specific group (community) that shares policies, requirements, concerns, values. Its management, ownership and operation is done by any combination of organizations in the community and a third-party [BGPCV12].

- **Hybrid cloud**: Composed by a combination of various cloud deployment models which have been already mentioned so that they remain unique but allow interchanging of data or applications. Cloud interoperability and standardizations are essential for this model.

Dillon et al. also mentions another deployment model, *Virtual Private Cloud*, developed by Amazon Web Services (AWS), which is between the public and the private solution and offers a protected link between the IT infrastructure of an organization and the public cloud of Amazon.

2.2 Cloud Applications

Applications based on the cloud are nowadays being popularly deployed and this should be done in the optimal way to be able to provide good services to the users. Studying their structure and individual components may help in that process. Modelling its application topology can be done through languages or standards such as GENTL, Blueprints or TOSCA.

2.2.1 Topologies

Applications tend to be complex and composed by various individual components that add functionality to it and are related among them. Portability regarding a cloud application is essential to avoid the lock-in effect and ease the deployment and management processes. This could be achieved by using portable, standardized and machine-readable format to model these components and the relationships established between them [BBKL14]. Some of the cloud application language topologies will be presented although there exist more (e.g. Blueprints, AWS CloudFormation, CAMP).

GENTL

GENTL stands for *Generalized Topology Language*. It models and supports the robust design of topology of applications based on the cloud and aims to reuse existing topology representations in a way that allows to extend them in the future and form complex models by combinations of other ones [ARSL14].

GENTL defines a metamodel whose basic component is the Topology element and contains a name and unique ID. From it, we can define the Component elements linked by Connectors (for a pair) that describe the relationship between them and whose type is represented by the Connector Classes. Moreover, components can be associated in Groups and form subtopologies. This topology language supports also defining Attribute elements for Topology, Component and Group items that can be of two types : Simple, consisting of a name and a value; and Composite, sequencing attributes to enable nesting [ARSL14]. GENTL provides

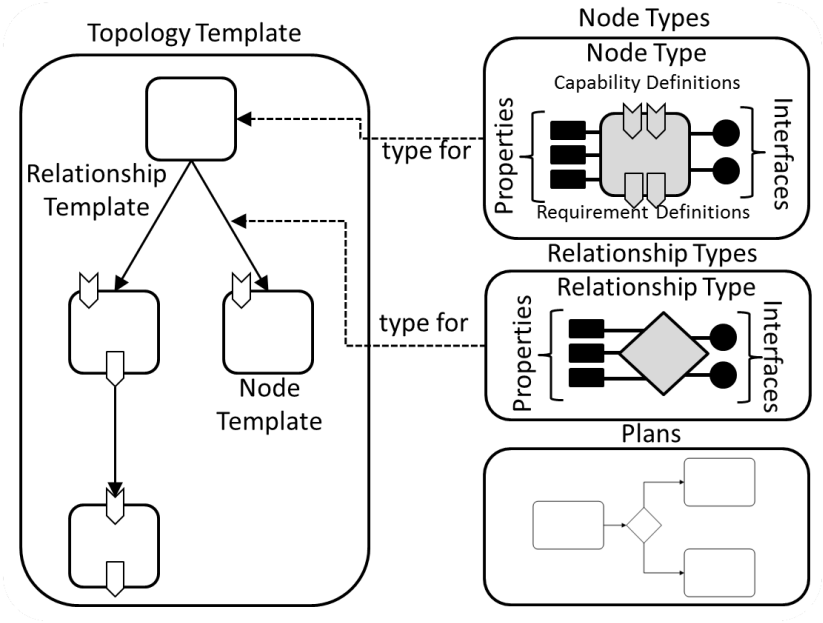

Figure 2.2: TOSCA Service Template[PS13]

also the possibility of adding additional (but not directly connected to the topology) information by using static or dynamic Annotations. They can be address to machine processing (automatic), human beings (human-oriented) or a combination of both (hybrid) [ARSL14].

TOSCA

TOSCA stands for *Topology and Orchestration Specification for Cloud Applications*. It is an OASIS standard that offers a standardized and definite language that describes the application components, relations and their management. It adds portability and automated management to applications and reusability to its components [BBKL14, BBLS12]. Its standarized container file type is CSAR (Cloud Service ARchive) with an organized structure [PS13].

TOSCA has two different parts [BBKL14, BBH+13]: Topology and management plans. The first element focuses on the components and the dependencies and relationships between them as well as describing its management capabilities that are used by the second one to allow the deployment, configuration, management and operation of the application through high-level manager tasks. The topology is composed by a graph with two elements: nodes (representing the components) and connections between them (representing the established relationships, e.g. 'installed on', 'deployed on', 'connects to') whereas management plans have an external message as the starting point and calls management operations related to the nodes, which depend on the type of elements used in the topology. The metamodel in TOSCA (defining the structure and management of the application) is composed by three conceptual layers both for nodes and relationships: Types, templates and instances [BBKL14].

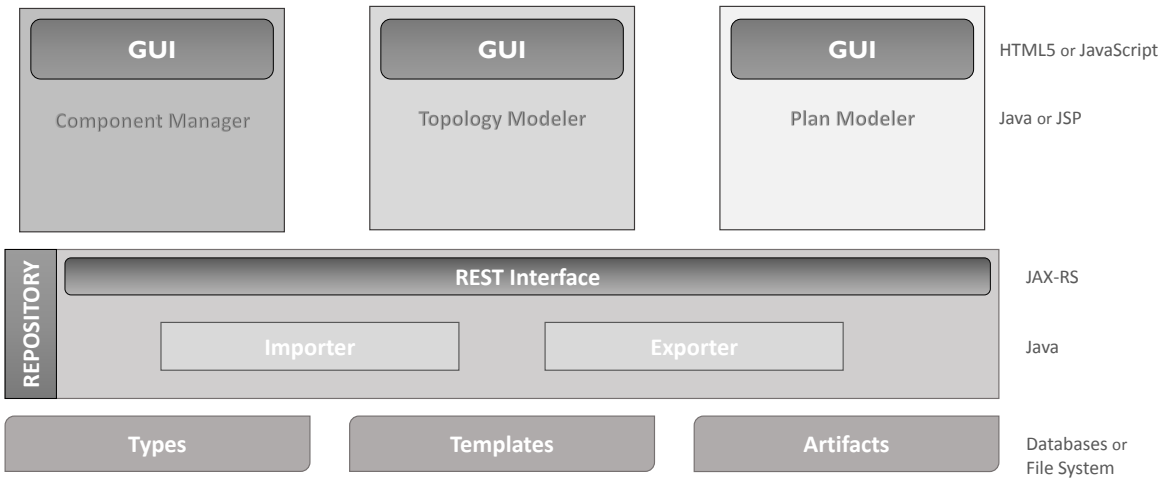

Figure 2.3: Winery Components Scheme

OpenTOSCA Winery

Winery is an open source modelling tool of cloud applications based on TOSCA standard, simplifying the development of CSARs through a web browser (i.e. Chrome and Firefox). It divides the TOSCA metamodel in two groups managed by: (1) the Topology Modeler and (2) the Element Manager, and stores all the data in (3) the Repository [KBBL13]. (1) allows a simple and visual graphical modelling of applications through a GUI whose element types can be configured or created using (2). However, Winery does not cover a TOSCA runtime environment and, therefore, does not support the instantiation of the application. For that purpose, OpenTOSCA could be used [KBBL13]. Additionally, there could exist a BPM4TOSCA plan modeller that supports the creation of BPMN models with required elements and structures by TOSCA plans. Figure 2.3 shows the general component organization[1].

2.3 Pearson's Correlation Coefficient

Pearson's correlation coefficient (r) represents the statistical linear relationship between two variables (X and Y). Its numerical values range between -1 and +1 and do not have units. This coefficient measures the type of relationship and its strength:

1. **To measure the type**: The sign of the coefficient shows the type of the relationship:
   - *Positive sign ($r > 0$)*, a direct relationship is established (i.e. when X is high, Y too and when X is low, Y is too).
   - *Negative sign ($r < 0$)*, an inverse relationship is established (i.e. when X is high, Y is low and vice versa).

[1] based on https://projects.eclipse.org/projects/soa.winery

Correlation	Interpretation
$0 \leq \|r\| \leq 0.2$	Very Weak
$0.201 \leq \|r\| \leq 0.4$	Weak
$0.401 \leq \|r\| \leq 0.6$	Moderate
$0.601 \leq \|r\| \leq 0.8$	Strong
$0.801 \leq \|r\| \leq 1$	Very Strong

Table 2.1: Pearson's coefficient interpretation [NST13]

2. **To measure the strength**: The intensity of the link is calculated with respect to the absolute value of r. The closer $|r|$ to 1, the stronger the relationship is. See table 2.1.

2.4 Utility Theory

Used in both economics and decision theory,[STFG08] defines *utility* as *'a value that represents the desirability of a particular state or outcome'*. [LL11] sees it as *'the satisfation that each choice provides to the decision maker* and for [Joh07] it is *'the perceived value of a good'*. Utility theory is a transversal area to mathematics, economics and psychology that studies quantitatively preferences under uncertainty [RYF15b]. It assumes that maximizing utility principle is the base for making decisions [LL11].

2.4.1 Overview

As [Joh07] explains, according to modern psychology, the perception of wealth is not fixed. The stated example is the difference in value that £1 would mean for a beggar and for a millionaire, as the wealth of the first one may highly increase whereas for the second one will not make any difference. Utility functions try to establish the relationship between the perceived and the real value and work over a non-empty and comparable set X of elements [Fis88].

The mostly used relation in utility theories, as well as in this thesis, is the asymmetric binary relation [Fis88] $\succ$ on X. With $x, y \in X$, $x \succ y$ indicates that x is preferred over y. Moreover, the indifference $\sim$ (no strict preference) and the preference-indifference relation $\succeq$ are defined.

Most utility theories assume the following four characteristics [Joh07, Fis88]:

Independence. Utility functions are independent between them as they are random variables.
Completeness. All possible outcomes are associated to a utility.
Transitive: If $x \succ y$ and $y \succ z$, then $x \succ z$.

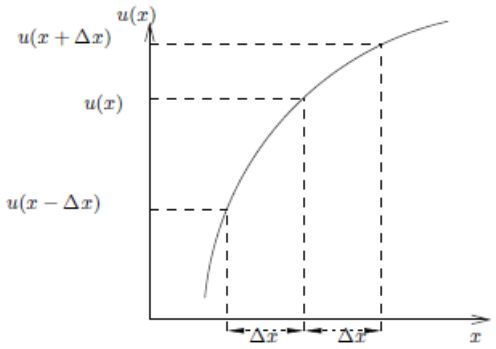

Figure 2.4: Utility Function Example [Joh07]

Continuity. When continuous wealth.
Utility are also increasing functions, fulfilling

$$u'(x) > 0$$

as more wealth is more desired. However, as in the beggar example, adding wealth has more value when the previous one was lower (decreasing marginal utility) [Joh07]. Figure 2.4 shows how the same increase in x ($\triangle x$) implies a lower amount of utility increase. Therefore (as risk-aversion characteristic),

$$u''(x) \leq 0$$

Moreover, if $\succ$ is assumed to be transitive and X is countable, there exists a real-valued function u on X satisfying:

$$x \succ y \Rightarrow u(x) > u(y) \qquad for \quad \forall x, y \in X$$

Some common and simple examples of utility functions are

$$u_1(x) = 1 - \exp^{-\alpha x} \qquad with \quad \alpha > 0$$

$$u_2(x) = \log x$$

But more complex utility functions can be generated depending on its use.

2.4.2 Economic applications

Utility has been used in many academic papers related to different topics such as scheduling batch tasks [IGC04], medical cases (as [PSW80], that studies the application or avoidance of prescribing coronary artery bypass surgery through utility theory) and of course in more economical challenges, as finding the optimal financing decision, consumption and investment strategies maximizing the consumption utility [Hak70]. More related to the topic of this document, utility functions were used in [WTKD04] to achieve the self-optimization in autonomic computing systems. However, an important use for utility theory was made in economics, for example in taxations [Sti50].

Risk-Aversion

From utility theory, decisions under uncertainty also consider the different attitudes by people towards risk. This can be expressed by the shape of the utility function. The fundamental concept of risk aversion is that the decision-maker would rather prefer the outcome of a non-degenerate lottery over the lottery itself [Bar07]. This is stated by Jenson's equality, with concave utility function, that fulfils for any random variable X that

$$\mathbb{E}[u(X)] \leq u(\mathbb{E}[u(X)])$$

Papers such as [RYF15a], [Bar07], and [ZL11] have followed this line and it is also used in investment decisions (e.g. [Tho03]).

Revenue and Cost

However, this paper aims to focus on the revenue and cost concept definitions through utility functions. One of the advantages this method offers is the possibility of combining different axes of interest into a single utility value.

Lin et al.[LL11] apply utility theory to a real case: maintenance and repair decisions regarding the pavement of a city of Taiwan. They aimed to optimize the decision of budget allocation combining different utility factors (such as pavement quality, people's opinion and traffic volume) and minimizing cost.

Strunk et al.[STFG08] study the case of using utility functions in a provisioning storage system situation. Then, the obtained results are used to improve the storage system configuration. The generalized form of the utility function is taken to be a sum of independent sub-expressions and more specifically, if it is related to revenue and cost, the following general expression is reached

$$Total\_Utility = Revenue - Cost$$

Therefore, the total utility of the system is found to be the subtraction between the revenue of the system and the cost associated to it (considering the cost as positive).

2.4.3 Utility Maximization as Optimization

Using a standardized topology language that offers portability between different cloud systems (e.g. TOSCA) will ease the deployment of applications. Moreover, finding their optimal distribution is essential to be able to reduce costs from execution and data transfers as the pay-per-use model is commonly applied in cloud computing and due to the vast amount of solutions available at the market, some challenges may arise.

Pandey et al. approach this topic focusing on reducing the total execution cost of applications by using the Particle Swarm Optimization (PSO). They study this meta-heuristic technique as the solution for dynamically schedule workflow applications [PWGB10].

Focusing on the migration configuration of an application to the cloud, if it has not been developed particularly for the cloud (cloud-native) but it should be adapted in some way to be suitable for it (cloud-enabled), an analysing and optimization of the chosen distribution is important [ABLS13]. Designers have to work both on how the distribution should be done and which solution of the market could lead to the best performance. In [ARXL14], the authors focus on cost-efficiency and propose a process consisting of four ordered steps:

(1) *Application modelling*, determine the topology of the application without taking into account any information regarding cloud offerings (it could be used an specific language for this, e.g. TOSCA, but also any other option that later on could be translated to one). (2) *Mapping to offerings*, possible offerings - and, therefore, possible providers- are selected through an identification process at the desired level (entire topology, sub-topology or component) using the application topology graph obtained after (1). (3) *Cost calculation*, obtains the cost of each of the possible offerings by using the result of the previous stage (called enriched topologies). The combination of the enriched topologies with the addition of the results of (3) leads to a group of cost-annotated topologies. The final step is the (4) *optimal topology selection* based on the results of (3), and choosing the best solution for a concrete usage profile. If different usage profiles are used, step (3) is repeated.

2.5 Python

Pyhton [2] is an open-source programming language (whose name is inspired in 'Monty Python') used throughout this thesis for different tasks such as data analysis and plots. Created in the early 1990s at Stichting Mathematisch Centrum in the Netherlands, this *high-level, interpreted, interactive and object-oriented* [3] language focuses on clear user syntax. One of its particularities is the strict role of indentation, as the interpreter of the code will divide it in blocks depending on the indentation. This is opposed to the more 'aesthetic' role that it has in other programming languages such as Java.

2.5.1 Standard Libraries

[Fou16a] collects the built-in C modules that form the standard library that Python software comes along with.

One of the contained modules is *csv*, that allows users to read and write in CSV (Comma Separated Values) format, which is one of the most common for databases. The reading method implemented in the module is *csv.reader(csvfile, **fmtparams* and it can be used after opening the csv file. For that, the *open(filename, mode)* is used. Required '*mode*' attribute is 'r' and 'w' for reading and writing modes respectively but they are not the only ones. Moreover, *writerows()* method is used to write a complete row. There exist many optional parameters for

[2] Reference manual at https://docs.python.org/3/reference/index.html
[3] https://docs.python.org/3/license.html

the *csv.reader()* and *csv.writer()* functions. Therefore, checking the documentation for more information is advised.

2.5.2 External Libraries

Python language also allows importing diverse packages and modules to be able to extend the resources and possible usable functions. Some examples are exposed:

SciPy

SciPy[4] is a Python group of open-source software for science, mathematics and engineering. Some of its core packages are SciPy library, IPython, Sympy, pandas, NumPy and Matplotlib. The last three of them were used in this thesis and, therefore, are explained in more detail afterwards.

One of the modules contained in scipy is *optimize* where the function $curve\_fit()$ is included. This function uses the non-linear least squares method to fit a defined function f to some data passed as a parameter. Its definition according to the documentation[5] is:

*scipy.optimize.curve_fit(f, xdata, ydata, p0=None, sigma=None, absolute_ sigma=False, check_ finite=True, bounds=(-inf, inf), method=None, jac=None, **kwargs)*

The main (and compulsory) parameters are:

f is the model function to fit the data to. It should be defined so the first argument corresponds to the independent variable and the parameters to fit as separate arguments.

$xdata$ sequence or array containing the data of the independent variable.

$ydata$ dependent data (*f(xdata,...)*.

Moreover, another important parameter of the function is $p0$. It represents the initial values for the parameters fitting. It is an optional parameter. If not specified, 1 is taken as initial value for all. This method returns:

$popt$ Array containing the guessed optimal values for the parameters.

$pcov$ Estimated covariance matrix of popt.

In this method, parameters are iterated and calculated to minimize the sum of the squared error of $f(xdata, *opt) - ydata$. One example of use when trying to fit some data to a cosine function would be:

[4] https://scipy.org
[5] http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_ fit.html

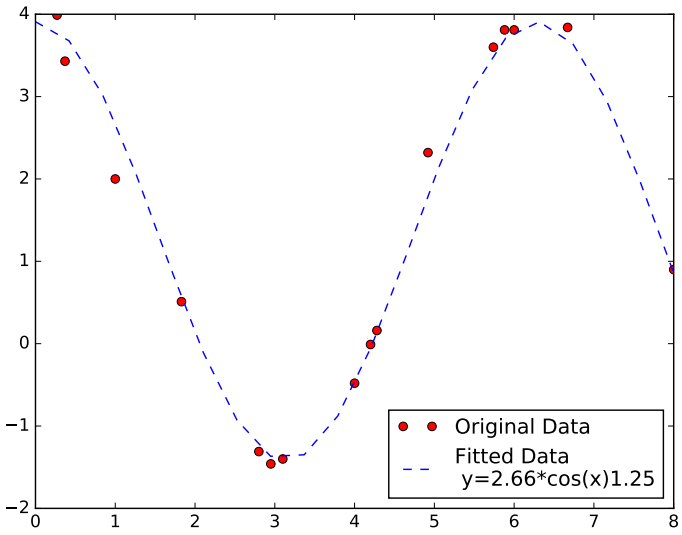

Figure 2.5: Curve Fitting Example

```
from scipy.optimize import curve_fit
from numpy import cos, linspace
import matplotlib.pylab as plt

def cosine(x, a, b):
    return a*cos(x)+b

xdata = [0.27, 0.37, 1, 1.83, 2.8, 2.95, 3.1, 4, 4.2, 4.28, 4.92, 5.74, 5.88, 6, 6.67, 8]
ydata = [3.99, 3.43, 2, 0.51, -1.31, -1.46, -1.4, -0.48, -0.01, 0.16, 2.32, 3.6, 3.81, 3.81, 3.84, 0.9]

popt, pcov = curve_fit(cosine, xdata, ydata, p0=(2.5, 1.2))

x_fitting = linspace(0, 8, 20)

plt.plot(xdata, ydata, 'ro', label='Original Data')
plt.plot(x_fitting, cosine(x_fitting, popt[0], popt[1]), '--',label='Fitted Data' + '\n y=' + str(round(popt[0], 2)) + '*cos(x)'+ str(round(popt[1], 2)))
plt.legend(loc='best')
plt.show()
```

Figure 2.5 shows the result of both the original data (red dots) and the fitted cosine function (blue lines). NumPy and Matplotlib libraries were also used in previous example. They are mentioned in sections 2.5.2 and 2.5.2 respectively.

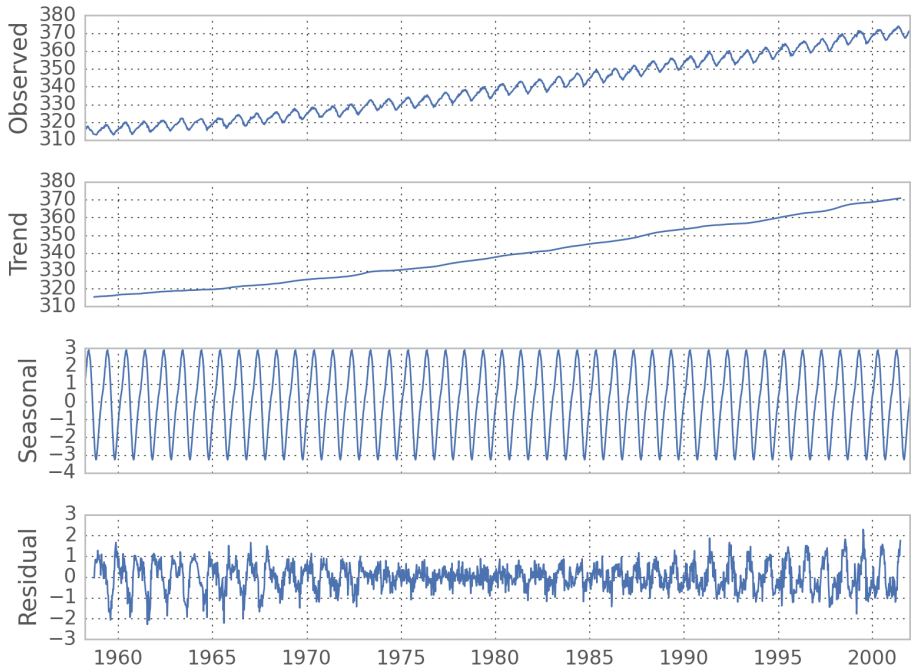

Figure 2.6: Seasonal Decomposition Plot Example

NumPy

This popular Python library for scientific computing[6]is fundamental because it provides functions and data structures whose high-performance is not provided with the standard library. One of the main features that this Python library offers is the '*powerful N-dimensional array object*' (ndarray) as well as operations on them in an efficient way. The ndarray is a multidimensional container of elements that are equal in type and size, that can be accessed and modified.

Statsmodels

Statsmodels[7] package comprises a set of statistical functions, based on R functionalities, that increases the possibilities of Python programming. Furthermore, combinations with other modules are possible (e.g. R writing style, ndarrays -see section 2.5.2- or DataFrames - 2.5.2- are supported).

Contained function *seasonal_decompose* (*statsmodels.tsa.seasonal.seasonal_decompose*) was used through this thesis and is therefore explained. Based on R function *decompose*, aims to split a seasonal time series function in three sub-functions. Figure 2.6 shows an example of the

[6]http://www.numpy.org/
[7]http://statsmodels.sourceforge.net/0.6.0/index.html

result. *Observed* function is decomposed into the sum of the *seasonal* component, the *trend* that the function follows and the *residual* function, so that the studied function is obtained.

Pandas

Pandas[8] is an open-source Python library that adds high-performance data analysis functionalities in Python, avoiding the necessity to change to another more specific programming language (e.g. R). It was developed on the basis of NumPy library.

One of its main elements is the creation of two new data structures: the Series and the DataFrame object. Its common characteristic is the allowance of integrated indexing but they also have a different point. The first one will be used as a one-dimensional structure for sequences whereas the second one is designed for more dimensions. Panda's DataFrame is similar to the ndarray structure but for example, each column could have a different data type whereas that is not possible with a ndarray.

An example is used to illustrate this element. The common creation of a DataFrame is done by passing to its constructor the required data. In the following example, that data is a set of columns composed by a key (*age*, *name* and *surname*) and some values. After the importation of pandas package, the structure can be generated just by calling *DataFrame* constructor and passing the desired data.

```
1    example_data = {'age':[18, 21, 30, 9], 'name':['John', 'Laura', 'Mark', '
                                        Anne'], 'surname':['Green', 'McKinsey',
                                         'Jones', 'Smith']}
2    data_fr = pd.DataFrame(example_data)
```

Then, the result of printing the object is represented in figure 2.7 (prints will represent the columns in alphabetically order -with respect to their labels- from left to right). It also shows the column and row indexation of this object. Another option when creating this structure is specifying which columns of the data element the user wants to include in the DataFrame. Figure 2.8 would be the obtained printed output.

```
1    data_fr = pd.DataFrame(example_data, columns=['age', 'surname'])
```

Moreover, any row indexation can be chosen if the user prefers it over the default one (from 0 to $N-1$, being N the number of rows). For that purpose, the optional attribute *index* of the constructor should be used. An example using ordinal numbers would be (see figure 2.9):

```
1    data_fr = pd.DataFrame(example_data, index=['first', 'second', 'third', '
                                        fourth'])
```

[8] http://pandas.pydata.org/

	age	name	surname
0	18	John	Green
1	21	Laura	McKinsey
2	30	Mark	Jones
3	9	Anne	Smith

Figure 2.7: DataFrame Example

	age	surname
0	18	Green
1	21	McKinsey
2	30	Jones
3	9	Smith

Figure 2.8: DataFrame Select Columns

	age	name	surname
first	18	John	Green
second	21	Laura	McKinsey
third	30	Mark	Jones
fourth	9	Anne	Smith

Figure 2.9: DataFrame Selecting Index

	name	surname
18	John	Green
21	Laura	McKinsey
30	Mark	Jones
9	Anne	Smith

Figure 2.10: DataFrame Indexed by Column

If we want to choose a specific column and index by it, we can do the following (figure 2.10):

```
1    data_fr = pd.DataFrame(example_data, index=example_data.get('age'),
                                          columns=['name', 'surname'])
```

Indexing by column is done in an intuitive way by using the column label as an attribute of the DataFrame object. The return object is a Series data type as seen in figure 2.11:

```
1    print(data_fr.age)
```

If a row or a set of rows wants to be selected, *ix* attribute is used with the row index. Example of the code and the result (figure 2.12):

```
1    print(data_fr.ix[1])
```

Panda's package also offers support for reading and writing with csv files. The basic reading method is *pandas.read_ csv(filepath_ or_ buffer, * * opt_ params)* and the corresponding writing one is *pandas.DataFrame.to_ csv(filepath_ or_ buffer, * * opt_ params)*. There exist the option of

```
0    18
1    21
2    30
3     9
Name: age, dtype: int64
```

Figure 2.11: DataFrame Column Indexation

```
age              21
name          Laura
surname    McKinsey
Name: 1, dtype: object
```

Figure 2.12: DataFrame Row

specifying a column as indexation for the DataFrame obtained when reading:

```
data_fr = pandas.read_csv('file-to-read.csv', index_col='Label-of-chosen-column'
```

Another used functionality offered by this method is date parsing to obtain datetime instances. For that, a date parser should be created depending on the receiving value structure, and adding two optional parameters to the function call:

```
data_fr = pd.read_csv('file-to-read.csv', parse_dates=True, date_parser=
created_dateparser, index_col='Label-of-date-column-to-parse')
```

If *parse_ dates*' value is 'True', index is parsed. However, there is also the option of specifying a set of columns to be parsed.

Matplotlib

Matplotlib[9] is an open-source Python-based 2D plotting package (including 3D charts). It eases the generation of various kinds of plots such as bar charts, histograms, polar charts or scatterplots [H+07]. With a philosophy of being able to create simple plots with a few commands, matplotlib has a vast range of plotting possibilities depending on the user requirements.

Pyplot is a sub-module included in matplotlib package that provides a framework so that its use is similar to MATLAB[10]. The basic functions of pyplot are:

1. *pyplot.figure()*: Creates a figure object.
2. *pyplot.plot()*: Plot required data. Options allow to personalize the visualization adding options such as choosing the color, width and type of used line.
3. *pyplot.title()*: Set a title for the plot.
4. *pyplot.xlabel()* and *pyplot.ylabel()*: Set x and y axis shown labels respectively.
5. *pyplot.xlim()* and *pyplot.ylim()*: Set or get x and y axis limits respectively.

[9] http://matplotlib.org/
[10] http://es.mathworks.com/products/matlab/

6. *pyplot.label()*: Displays the label of each plot with the corresponding text as defined by the user in the arguments of *pyplot.plot()* function.
7. *pyplot.show()*: Displays the figure(s).

2.6 Kereta

Kereta is a framework developed by Frech in [Fre16] that can be used as a supporting tool to perform the utility-based evaluation.

2.6.1 Overview

The growing popularity of Cloud Computing and its applications has motivated an increase in the number of market offerings related to it. Some well-known companies that have taken part in the cloud business are Amazon, Google, IBM and Red Hat. However, the wide range of possible cloud service and deployment models (see sections 2.1.2 and 2.1.3 respectively for more reference on this) together with the high amount of organizations in the sector have led to a vast solution spectrum. Users facing cloud computing 'world' that want to deploy an application on the cloud are challenged to find the optimal topology solution balancing costs and performance.

That is the motivation of [Fre16]. His approach focuses on the evaluation of the different deployment topology options for applications on the Cloud, following a utility-based study method. As a result, a decision support framework, Kereta, was designed.

2.6.2 Utility Function

[Fre16] follows a utility-based analysis aiming to evaluate different topology deployments for business applications on the cloud. It defines the utility function U as:

$$U(T^i_\mu, W, R, t) = rev_{exp} - cost$$

where:

t is a definite time interval,

T^i_μ refers to the μ-topology under evaluation,

W is a set of m application workloads at T,

R is a set of requirements $r_0, r_1, ...r_n \in R$,

rev_{exp} measures the expected revenue of the application and

$cost$ measures the costs of the distribution.

Term rev_{exp} is calculated as the definite integral of the revenue per time unit on interval t:

$$rev_{exp} = \int_{t_{min}}^{t_{max}} rev_{<unit>} dt$$

where

$$rev_{<unit>} = \sum_{j=1}^{m} p(w_j, t) * user\_av(t) * tpu(w_j) * rpu(t) * sat(T_\mu^i, t) * availab(T_\mu^i, t)$$

p is the probability of receiving a concrete workload w_j at t,

$user\_av$ is the average number of users at t,

tpu represents the average number of transactions per user,

rpu measures the average application revenue per user,

sat is the average user satisfaction of T_μ^i at t and

$availab$ values the availability of T_μ^i at t.

Term $cost$ is calculated as the sum of costs of a concrete distribution T_μ and the adaptation costs to ensure satisfying the set of requirements R at time subintervals $\{t_0, t_1, ..., t_k\}$, originating topology $T_\mu^{i,j}$:

$$cost = cost_{fixed}(T_\mu^i, t) + \sum_{q=0}^{k} cost_{adaptation}(T_\mu^{i,q}, W, R, t_q)$$

2.6.3 Architecture

The system developed by Frech follows the scheme of figure 2.13. As it can be seen, Kereta module is composed by three elements: (1) a repository, (2) a calculation element and (3) a MySQL database enabled through a JDBC (Java DataBase Conector driver), as well as an interface between the module and the previous Winery modeling tool ([KBBL13]).

Supported resources, explained in 2.6.4, are managed through the repository and are used afterwards by the calculation component in order to perform the computation of the defined utility. Both elements can be accessed through a REST-API interface. It adds the HTTP methods (GET, PUT, POST and DELETE) and return a status code with an XML representation in case an error occurred.

On the other hand, the database is used to persist resources by using different tables. Each entry is composed by fields, its corresponding data type and description. Some resources have also a key (that can be private or foreign).

Moreover, a suitable GUI to Kereta is provided by the extension of Winery modelling tool.

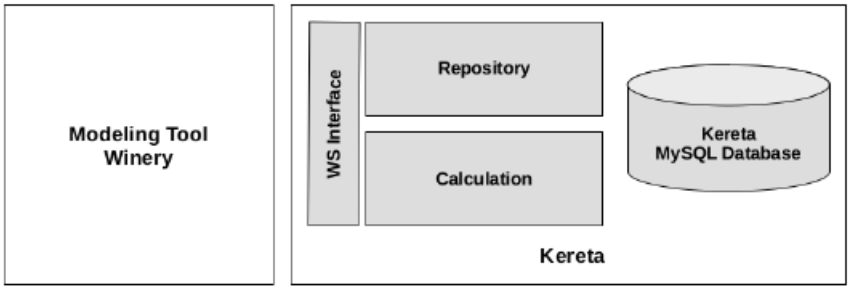

Figure 2.13: System Design Scheme [Fre16]

2.6.4 Offerings

Kereta's repository allows management of resources with a REST-API. Implemented resources are divided in five groups each one containing one or more elements: (a) Application, containing application-specific topology, application-specific components and requirement resources. (b) Distribution, composed by application distribution, application sub-graph and performance fields. (c) Utility function, including themselves and its sub-functions. (d) Function, constituted by function and parameter resources. And (e) type, covering application type, requirement type, function type and data type.

Users can make use of HTTP-methods through the API for each one of the previously mentioned resources (obtaining as a result a XML-document of messages or resource 's representation when needed): GET (returns a group of resources or the outcome value of a function), PUT (is not allowed to change resource properties related to their URIs), POST (creates a resource regardless of its level as it can be rooted or nested) and DELETE (eliminates single resources - and its nested ones). Moreover, the repository implements a search function, allowing the reusability of functions and utility functions (which can also be cloned) and the evaluation of the previous mentioned resources together with sub-functions by key assignments specified at the URI, that will be passed to the calculation module. In it, function formula is transformed to a processable tree representation by reverse polish notation method (reached through Shunting-Yard Algorithm) and the result is calculated.

The calculation module supports complex formulas, that can be composed by: sums, subtractions, multiplications, divisions, exponents, integrals, roots, factorials or boolean operators (not- $\neg$ -, or- $\vee$ -, xor -$\oplus$-, and -& -, lower and bigger than -< and > respectively-, equal -=-, and the combination of these last pair of operators -$\leq$, $\geq$ -). What is more, sine, cosine and tangential functions can be used, as well as minimum and maximum comparison.

Moreover, the parser is also able to manage if and else statements, parenthesis (of both types -() and []) and e and π constants.

3 Related Works

3.1 Utility-based Analysis in Cloud Computing

Utility-functions have been widely used in cloud computing analysis: for instance, to forecast possible workload to ease optimal resource auto-scaling [RDG11], to help calculating possible guaranteed Quality of Services (QoS) in cloud environments [XP09] or related to mapping cloud resources to workflows [PDAL+09].

Paton et al. aim to maximize the utility, which they define in two ways. Related to response time:

$$Utility_w^{RT}(w,a) = \frac{1}{\sum_{i \in w} PRT_w(i, a_i)}$$

And related to profit:

$$Utility_w^{Profit}(w,a) = \sum_{i \in w}(Income(i,a_i) - EvaluationCost(i,a_i))$$

In these formulas, w is the group of workflows, a are the assignments for instances i in w, a_i is the workflow assigned to instance i and the predicted response time of the assigned workflow is PRT_w. Moreover, evaluation of instance i with assignment a_i leads to assets received due to it (*Income*) -that also depends on PRT_w - and costs of the resources used in the evaluation (*EvaluationCost*). This last definition of utility function is similar to the one applied at this thesis, as a trade-off between revenue (income) and cost (evaluation cost), although [PDAL+09] analysis is based on workflows.

Minarolli and Freisleben' approach [MF11] also proposed a maximization of utility functions, in this case, with the goal of finding the optimal resource allocation for virtual machines under an *Infrastructure as a Service* model (see section 2.1.2). This is done by the *local node controller*. The defined theoretical utility function from a virtual machine i is a linear function directly proportional to the paid money per CPU allocated resource unit (α_i, measured in \$/%) and the CPU percentage of allocated resources (called S_i by the authors):

$$u_i = \alpha_i * S_i$$

However, in practice they had to adjust this formula and add a term SO_i referencing the share of CPU needed for the implemented local node controller, resulting in the formula:

$$u_i = \alpha_i * (S_i + SO_i)$$

They also defined a utility function (N_j) per node j (n in total) as the sum of utility functions

of each virtual machine allocated on the node (m in total) minus the cost of the node (C_j):

$$N_j = \sum_{n=1}^{m} (U_{jn}) - C_j$$

The total utility function of the system (that should be maximized) then is given by the sum of nodes' utilities:

$$u_{total} = \sum_{t=1}^{n} N_t$$

Or, written using the cost-performance previous relation:

$$u_{total} = \sum_{t=1}^{n} \sum_{s=1}^{s=m} (U_{ts}) - C_t$$

On [ZWS06], Zhu et al. also use utility functions but in this case to express the desire of some QoS metrics y. The utility function of a service $h(y)$ is defined as a trade-off between the payoff in a service level agreement ($z(y)$)and the cost function ($c(y)$):

$$h(y) = z(y) - c(y)$$

Approach in [Fre16] used this functions to allow analysis of cloud distributions, resulting in the implementation of Kereta framework increasing the analysis in revenue and cost. This document defines the utility as a balance between the expected revenue (studying the trends of last years) and the cost derived of the cloud migration, both considering direct and adaptation costs. One advanced step is that not only the strict revenue is considered but also some possible influences (such as user satisfaction) are added to the analysis. However, in this thesis the definition of the terms aimed to be defined concretely and linked to real data measurements.

3.2 Web Application and Wikis Analysis

Web applications are constantly under development and its complexity has increased significantly as the number of available tools and resources for application developers rises. Moreover, improvements are highly valued and testing is one of the methods that can be used for it. [MJ98] already underlined the importance of making performance analysis but also its arduousness. [BMT05] agrees with that fact and states some value-add elements for web applications such as speed (updates and technology improvements), complexity and a mature design.

However, other area of testing and study closer to this thesis is the one that aims to find the optimal application distribution (regardless if it is for performance or economical reasons). For example, well-known Amazon started as a rigid application that ran only on one server. Nevertheless, the growth on popularity led to manageability problems and they ended up

changing its architecture into a distributed and decentralized system [Gra06] because of scaling advantages and easy service introduction.

[UPVS07] is also developed in this area but focuses its study on large-scale wiki sites like Wikipedia. It remarks the advantages of distributed systems and propose a new architecture that involves computer resources of supporters of this wiki, willing to host part of Wikipedia and becoming collaborative nodes (other distribution strategies for wikis have also been studied, e.g. peer-to-peer [Mor07]). The load is balanced on all of them although each page is located only on a single node in a dynamic way so its location can change. This leads to a redirection routing mechanism to the correct node that is managed by distributed hash tables. Moreover, this collaborative and distributed organization also has drawbacks and risks that should be faced: node failures -and recoveries- (could damage the integrity of the complete system) and security (guaranteeing that attacks from not reliable nodes are minimized). A method to evaluate the page distribution according to the load handling is requested by [UPVS07] approach. Assuming only limitations on disk space and bandwidth, the cost function reached is:

$$c(N,P,W) = \sum_{p \in P} \left[\alpha \left(\frac{i(p,W)}{i_{tot}(N,W)} \right)^j + \beta \left(\frac{o(p,W)}{o_{tot}(N,W)} \right)^j \right]$$

$i(p,W)$ represents in bytes the size of incoming requests for an specific page p during time window W and the maximum able to receive is $i_{tot}(p,W)$. The same reasoning follows $o(p,W)$ and $o_{tot}(p,W)$ but for outgoing bytes due to requests of page p in window W. α and β constants are used to weight the effect of each term and j is a constant greater than 1. It is directly proportional to the number and usage of resources needed by pages hosted in a node but inversely proportional to the resources provided and user's noticed performance. Urdaneta et al. also consider constant load measuring at the nodes to evaluate if a page movement is necessary.

Another study related to Wikipedia analysis is [UPVS09]. This approach adds realistic workload at the evaluation and states the importance of a good combination of aspects including distribution and replication (i.e. popular articles should be replicated in more servers as they are more likely to be read and updated). The data used through [UPVS09] is a trace of 107 days (between years 2007 and 2008) which contains a sample (10%, 20.6 billion HTTP requests) of Wikipedia's real traffic. After noticing that almost a 50% (45.05) of the previous workload was addressed to the English version of Wikipedia (en.wikipedia.com), the authors decided to evaluate a set of workload data of this wiki too. In addition to the address difference, the general analysis includes requests of all types (i.e. page creations, editions or reads; uploaded media; static file; cache maintenance; etc.) The results show that the three most frequent type of requests are addressed to: (1)static files, with a 24.04%; (2)uploaded media files, with a 21.88%; and (3)image thumbnails, with a 18.7%. [UPVS09] states that this types are the most popular due to the fact that they are usually embedded in wiki pages and, therefore, requested when its reading is performed. Old versions of Wikipedia pages are accessed with a frequency of only a 6% and [UPVS09] recommends storing them in a separate document from the last version due to the low frequency access so they can be located in nodes having low bandwidth. The authors also classified the pages according

to their managed workload (and, consequently popularity), offering different strategies (i.e. replication or load balancing) for each group.

3.2.1 WikiBench

Benchmarks are used also in researches on how to improve web application hosting systems. One example is RUBBoS. However, scalability is not achived as it can only be run on one system. To avoid that, and other limits of similar tools, WikiBench[1] was created. It is a developed software at VU (Vrije Universiteit) University of Amsterdam by Guillaume Pierre and based on MediaWiki that aims to be a testing tool for distributed web applications. Using real (although anonymized) Wikipedia access traces, it is able to generate close-to-reality workload with scaling advantages that other benchmarks such as TCP-W or RUBiS (auction-based) cannot offer. As [vB09] explains, apart from WikiBench application itself, this software is constituted also by TraceBench (which is in charge of trace processing) and post-processing tools (i.e. scripts responsible for analysing log files of the application and represent the obtained results).

The needed meeting requirements of WikiBench are: (1) realistic generated workload, (2) no-benchmark-size limits on its use, (3) consistent results when reproducing, (4) scaling ability, (5) configurable traffic settings and (6) portability and package-based (i.e. containing needed snapshots). An overview of its design is seen at figure 3.1. There exist three different areas: a controller, a worker node (or a set of them) and the tested system. The first one, after TraceReader processing (i.e. reducing traffic intensity but keeping characteristics such as reading/editing page ratio), organizes the distribution of work amount (POST and GET requests) of each node and coordinates the system through plain text messages. In addition, traces are sampled randomly but maintaining the realism.

This software can be useful in various computing research fields such as performance or capacity planning analysis.

3.3 Conclusion

There are studies about multiple ways of testing and distributing an application in a cloud system but the advantages of utility theory give another approach when considering different distributions for applications. WikiBench tool has many pros but the most significant one is its ability to offer testing with real data from Wikipedia, increasing the accuracy of the results, which can be used to improve the systems.

Moreover, utility theory offers the possibility of evaluating diverse metrics and adds flexibility as its design depends on the elements to maximize. Therefore, multiple definitions can be made depending on the desired evaluation. Basing it on the economical perspective, which is in fact the main objective in many cases, but taking into account other elements which

[1] http://www.wikibench.eu/

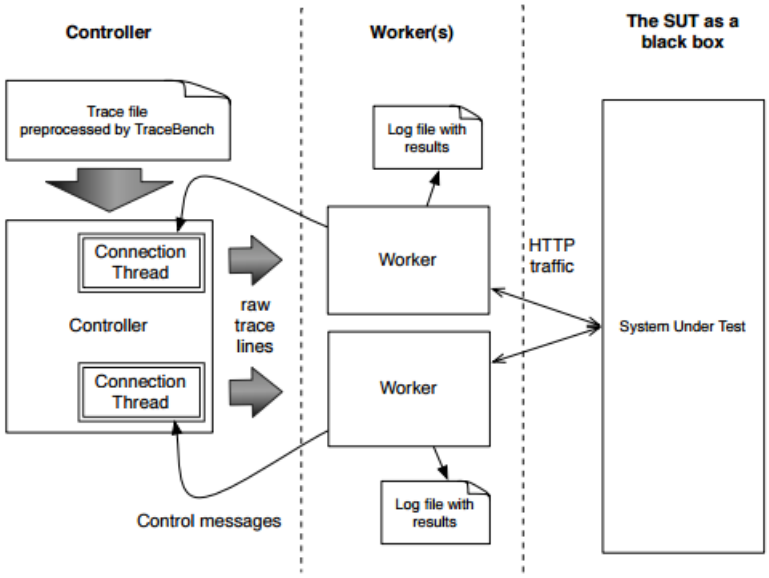

Figure 3.1: WikiBench Design Scheme [vB09]

can influence the final result, allows deepening at the study. It can rely on already existing tools, such as Nefolog (for cost calculation of different deployment configurations)[XA+13] or Kereta (for utility calculation), easing the process. Furthermore, a case study of a real web application can improve and support forwarding research on this area.

4 Business Application Revenue Models

In general, businesses and organizations' abstract methodology consists of producing an output with an input in a way that this process generates some benefits (i.e. the revenue overtakes the costs of the process) and meet customers' needs as shown in figure 4.1. The following formula summarizes the conceptual idea:

$$Net\_income = Revenue - Cost$$

4.1 Concept

A *revenue model* denotes a plan on how an organization will generate profits, earn income and get a higher return on investment, by making the most of its sources. They are considered essential for good business organization, helping to determine user segments and value their own potential.

There exist a wide range of models although the target sector, the type of company and corporate culture and policies are important when deciding which revenue model to follow. Lifetime of enterprises can also be relevant in that decision. For instance, mainly start-ups' revenue models are created in a long-term basis (i.e. the time plan is longer than one year) as their first objective goes towards a growth instead of generating revenue. One good example of this is the well-known Google, which waited two years after its company's creation before having plain text advertisements on the webpage showing user's search results.

4.2 Examples

Some examples of revenue models used nowadays in web applications are:

Advertising

This is one of the most common revenue models for web applications. The process consists of charging other companies for hosting and promoting their advertisements. There exist different types. The first one to appear is *cost per impression* (CPI). In this case, revenue depends on the times an ad is displayed. Generally:

$$Revenue = \sum_{n=1}^{N} \tau_n * \delta_n$$

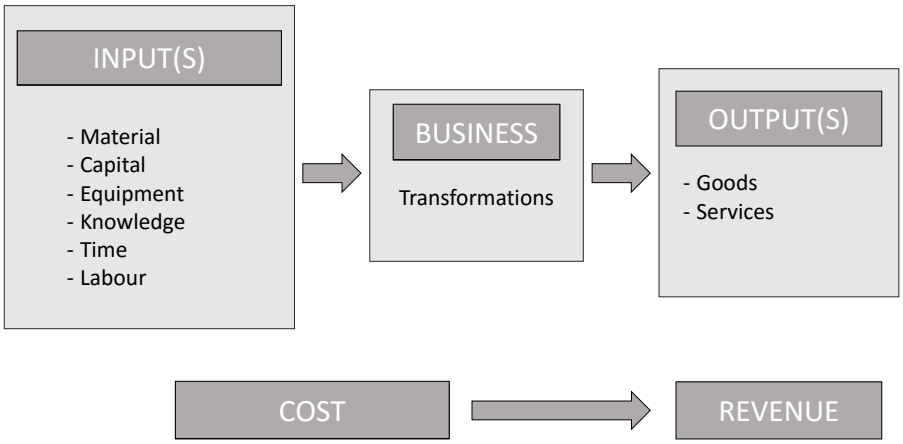

Figure 4.1: Business Process

where N is the total number of ads displayed, τ is the cost per impression and δ_n the total number of impressions for ad n. There exists also a version called *cost per mile* (CPM), whose reasoning is the same as previous CPI but calculated per a thousand impressions.

There is a variant of this model called *cost per click* (CPC), where the company owning the ads will pay the one hosting them depending on the number of times a user clicked on the advertisement. Therefore, another way of expressing it:

$$Revenue = \sum_{n=1}^{N} bid_n * \eta$$

where N is the number of total advertisements hold on a web, bid_n the fixed amount of money per click agreed for ad n and η the number of clicks on that advert. Well-known *Google*, through *AdWords*[1] follow this model. However, as [Eva08] explains, the basic fee in both options (τ and *bid*) depends on many factors such as:

- **Size**: Dimensions of the ad can also have an influence on the price.
- **Location**: Depending on the place where the ad is displayed. More visible ones would be charged higher.
- **Number of visitors**: Popularity of a page can cause changes on the fee (i.e. more visitors increase the basic price).

User subscription

In this model, revenue is obtained through a (usually monthly) charge in exchange of the possibility of using the web application. There exist some variants in this model that may

[1] https://www.google.es/adwords/

change the subscription price but the main characteristic is that users are, independently of the amount of money, always charged for the service. The most common are:

1. **Fixed subscription**: The subscription fee is fixed and the same for all users. One example of this method would be *SeeSo*[2], a web application hosting comedy videos, with a prearranged subscription charge for its users. If the web application has η users, revenue is stated as:
$$Revenue = fee * \eta$$

2. **Variable subscription**: There exist some subscription packages, associated to different features (higher fee is linked to more exclusive ones) such as higher service speed, time duration, number of available options or higher storage capacity. One example of this model would be *GraphicStock*[3], that offers three plans (monthly, annual and premium) which are differentiated among them in terms of time-limited duration of the subscription and the range of options available to the user. Revenue can be calculated as:
$$Revenue = \sum_{n=1}^{N} price_n * \eta_n$$
where N is the number of total subscription plans available at the application, $price_n$ is the price of package n and η_n the number of users that chose option n.

3. **A la carte subscription**: Each feature or option of the web application has its independent price and user is charged for the ones chosen, having the opportunity of a completely personalized configuration and a wide range of possible combinations. One example of this model would be *CreativeMarket*[4], where users can download design content from many artist, paying per download its associated amount. The general formula would be:
$$Revenue = \sum_{p=1}^{P} price_p * \eta_p$$
where P is the total number of products or features offered, $price_p$ is the cost of product p and η_p the number of users that have bought that characteristic, service or product.

Transaction fees

Revenue is commission-based, obtained for the operator through enabling or executing some transactions (e.g. selling a product). This operator offers another companies a platform to perform their transactions. That companies act as the same time as clients (of the market provider) and sellers (regarding general public). *Amazon* and *eBay* are examples of this model [Rap03].

[2]https://www.seeso.com/
[3]https://www.graphicstock.com/join/?utm_source=GAWGS-Search-Brand-DE-99&utm_-medium=cpc&utm_campaign=GAWGS-Search-Brand-DE-99
[4]https://creativemarket.com/

Revenue is usually both fixed and percentage-based. A fixed fee is charged for posting a product to sell and also a percentage of each sale should also be paid to the operator. Summarising the concept:

$$Revenue = \sum_{n=1}^{N} fixed\_charge + percent\_charge_n$$

$$percent\_charge_n = \alpha_n * \tau_n * \eta$$

where N is the total number of elements, α_n represents the agreed percentage for item n, τ_n is the price of element n, and η is the number of products n sold through the operator.

Freemium

Dropbox[5] is a good example of this technique. The word *freemium* comes from the union of *free* and *premium*. Some basic services are offered for free but users are charged for additional features. Dropbox, for example, offers more storage capacity - among other features - for users paying a fixed amount of money per month. A common variant of this revenue model is offering a free service with advertisements and the option of getting rid of them by paying some amount of money.

In this model, the sum of all exclusive options produces the revenue:

$$Revenue = \sum_{n=1}^{N} \alpha_n * \eta_n$$

where N is the number of total charged options, α_n is the fee that users who want the advance features of option n have to pay and η_n the number of users that took option n. In this model, if only one 'premium' option is offered, revenue can be simplified as:

$$Revenue = \alpha * \eta$$

Sponsorship

In this model, a company (or more) is charged with a fee in return of some sponsorship offerings such as standout advertisements, agreements regarding data licensing, priority in search listings or branding features (e.g. displaying their logo). For example, *Twitter*[6] offers preference in user listings for sponsor companies. In sponsorship, revenue depends on the amount of money and offerings agreed between the company and its sponsor.

[5] https://www.dropbox.com/plans
[6] https://business.twitter.com/es/help/overview/ads-pricing.html

Sales

Main income goal comes from selling a product, which can either be a good or a service. Although it was usually linked to a physical selling place, Internet managed to disassociate those concepts. The total revenue would be the total money earned by this and it would be calculated as the product between its price and the number of sold ones:

$$Revenue = \sum_{n=1}^{N} \tau_n * \eta_n$$

where N is the total number of products at that web, τ_n the money obtained through the sale of one product n and η_n the number of items corresponding to that product sold.

For example, *E-tickets*[7] allows users to create sales pages and gets some fixed money for each ticket sold.

Selling data

Digital area has changed significantly the traditional business models. Users are connected to the Internet and sharing constant data publicly. Many companies have realised how valuable personal data is. It can help to predict trends, users' likes and dislikes, understand their reactions to new products and increase knowledge about a targeted sector. The sentence *"if you are not paying for a product, you are not a customer, you are the product"* states this revenue model clearly. Personal data is worthy and many companies are willing to buy it while others have access to their users' data and know its market possibilities. It can also be seen as a special version of previous revenue model.

BDEX[8] for example, is a marketplace that joins companies using this revenue model (willing to sell data) and companies willing to process and use it (e.g. for targeting campains).

Another model is *donations-based*. Developed more in detail in section 5.1. It is important to point out that all these revenue models are not exclusive and two or more can be used by the same web application at the same time. The total revenue then is obtained by the sum of all types of revenue (being R the total amount of types used):

$$Total\_revenue = \sum_{r=1}^{R} Revenue_r$$

Table 4.1 shows the revenue models commonly used for some examples of applications type.

[7] https://www.etickets.to/
[8] http://www.bigdataexchange.com/

Web application type	Revenue model							
	Advertising	User subscription	Transaction fee	Freemium	Sponsorship	Sales	Selling data	Donations
E-commerce (ex. *Amazon*)	✓	✓	✓		✓	✓	✓	
Educational (ex. *Quizizz*)	✓	✓	✓	✓	✓			✓
MediaWiki (Wikipedia)								✓
CRM app (ex. *Salesforce*)		✓		✓				
ERP (ex. *Odoo*)		✓		✓				✓
Search engine (ex. *Bing*)	✓				✓		✓	
Email (ex. *Outlook*)	✓	✓		✓			✓	
Social network (ex. *Facebook*)	✓			✓	✓	✓	✓	
Language app (ex. *PONS*)	✓	✓		✓	✓		✓	✓
Online media (ex. *Youtube*)	✓	✓		✓	✓		✓	
CMS (ex. *WordPress*)	✓	✓		✓	✓			✓
Mapping (ex. *Google Maps*)	✓				✓		✓	
Document (ex. *Word Web App*) or file management (ex. *Dropbox*)	✓	✓		✓	✓		✓	
Chatting (ex. *HipChat*) or video calls (ex. *Skype*)	✓	✓		✓		✓		

Table 4.1: Common revenue models per application type

5 Wikipedia Revenue Analysis

Regarding wikis (database or website developed by a user community, allowing editions by any user), the study was focused on Wikimedia Foundation as it is one well-known organization that supports '*free knowledge projects*' [Fou16b] (i.e Wikipedia, Wiktionary, Wikiquote, Wikibooks, Wikisource, Wikimedia Commons, Wikispecies, Wikinews, Wikiversity, Wikidata, Wikivoyage and MediaWiki) and it has some open financial information [KPM, Fouc].

In the previous chapter, general revenue models were explained but this chapter focuses on Wikipedia, as the first step addressed to the utility-based analysis. First of all, an overview of MediaWiki and its topology is presented as it is needed for the analysis. After that, its business model and results of last financial years (2013-2014 and 2014-2015) are presented as the basis for its type definition. Data is then collected to study any possible strong relationship between the direct profits obtained and some statistical metrics (e.g. users, page edits) to analyse its inclusion in the revenue and utility model used in the document. Finally, revenue models are derived for the first semester of the year and are used in the performed evaluation at chapter 7.

5.1 MediaWiki Engine

MediaWiki is an open-source PHP-based software developed for wikis in 2002, relaying on a back-end database (e.g. MySQL). For that reason, easy scalability and security handling are important. Extensions, multimedia, mobile applications and user customizations are supported. It was initially designed to cover the needs of Wikipedia, but nowadays other well-known wiki projects, such as Wikimedia Commons or WikiLeaks, also use this software.

5.1.1 Topology

When considering the migration of an already-existing application to the cloud, approaches usually tend to consider *the whole software stack*, packaging it into virtual machines (VMs) running on the cloud (called type III migration in [ABLS13]). However, considering the application topology (i.e. representation of components of an application in middleware solution) as the basis for finding the optimal cloud distribution with less constraints could add more options such as multiple cloud provider offerings. For example, TOSCA standard (see section 2.2.1) represents the topology of an application as a labelled graph composed by nodes, edges and labels.

[ASLW14] defines three topology types for web applications. The α-topology is constituted by the needed application specific components, whereas the elements that are not specific or dependent of the studied application (and are, therefore, reusable for others) are called

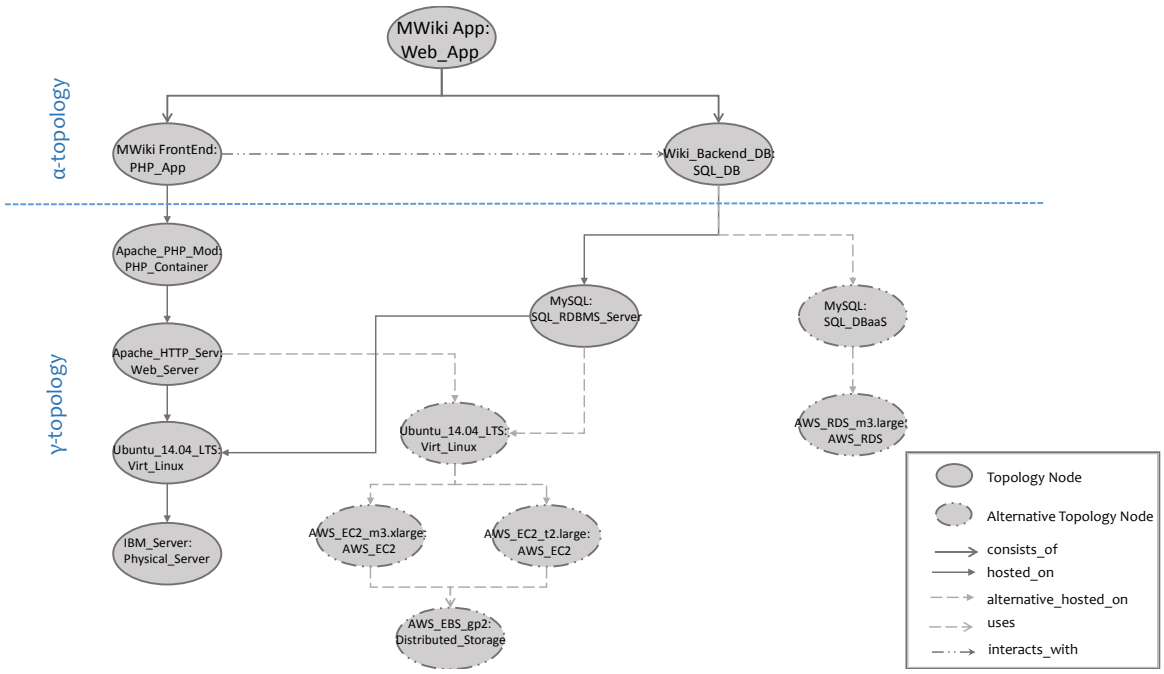

Figure 5.1: MediaWiki Topology. Based on [SALS15]

γ-topology. Both sub-topologies compose the μ-topology, which represents a viable topology for the application [ASLW14].

In this case, MediaWiki topology, represented in figure 5.1, is the basis under study for the distribution analysis. It is composed by two tiers: a front-end and a back-end database. These elements model the α-topology whereas the low part represents the γ-topology. PHP files deployed in a PHP container constitute the front-end. This container is developed as an Apache Server element running on a Linux virtual machine using Ubuntu, which is installed on an IBM server. The back-end also uses that virtual machine through a MySQL RDB server for the wiki database. The topology also shows some other alternatives for the used topology such as using a MySQL DBaaS solution deployed in AWS RDS database instance for the back-end.

5.2 Revenue Model

Wikimedia Foundation does not consider any of the revenue models explained before (see section 4.2) although some studies show that, for instance, advertising is an important online revenue source [Eva08]. It has a strong policy regarding advertisements which are not considered a way of revenue for the foundation as they *'do not believe that advertising belongs in a project devoted to free, reliable, and neutral knowledge. Introducing commercial interests could jeopardize Wikipedia's reliability as a neutral source of information'*. As discussed in [Yam13], there was a conflict between the non-profit role (editors as volunteers) and the necessity of raising capital (implying commercial interests), where protests held by some editors and the

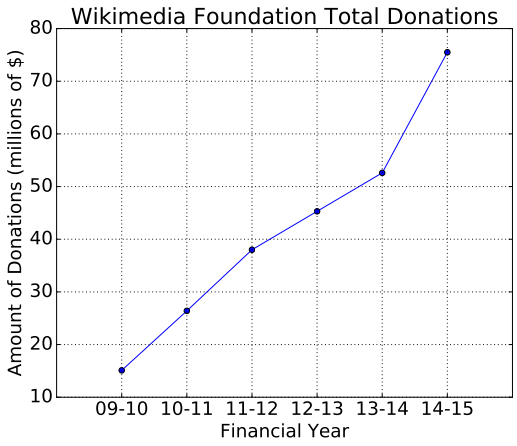

Figure 5.2: Wikimedia Foundation Total Donations

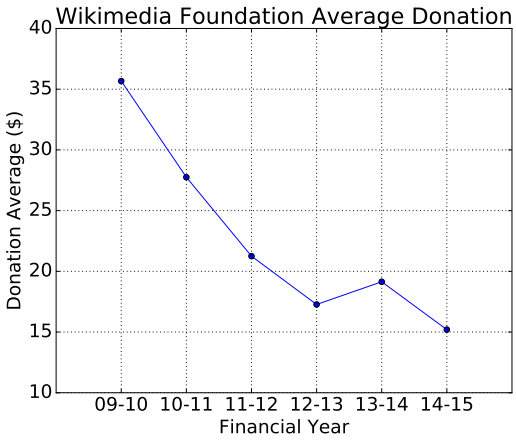

Figure 5.3: Wikimedia Foundation Average Donation

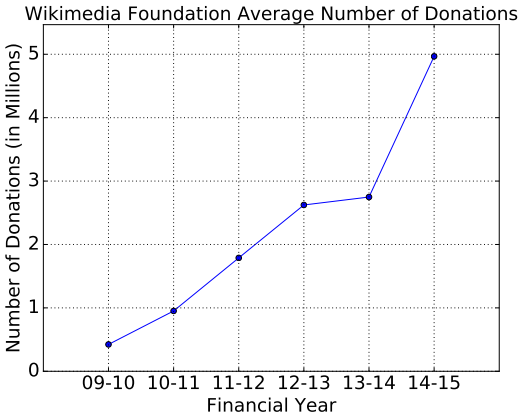

Figure 5.4: Wikim. Foundation Average Number of Donations

desire of neutrality led to the previously mentioned ads policy. However, [Yam13] also gives the example of Google search engine which, mantaining to be neutral in their commercial services, displays advertisements in its result page.

Fundraising/Donations. Wikimedia Foundation is a donation-based organization, received from users or companies (minimum donation is $1 for security reasons, as smaller donations are usually used by dishonest entities in order to try to get credit information). Benefactors (i.e. Peter Baldwin) and foundations (i.e. Alfred P. Sloan Foundation) [Foua] have also helped financially. In last published report [KPM], it is reflected that $72,236,884 of the total unrestricted net assets were collected through fundraising, which represents a 95.301%.

Figure 5.2 shows the ascending trend for donations on the last past years (from financial year 2009-2010 to 2014-2015), excluding commitments for future ones. In financial year 2009-2010 (i.e. from 1 July to 30 June of following year) the total donations received by the Foundation were around $\$15.1M$, increasing significantly ending in $\$75.5M$ in 2014-2015. That means an

increase of 400% in 5 years. It is shown that the maximum difference was between financial years 2013-2014 and 2014-2015.

At the same time, the average amount per donation is decreasing as seen in figure 5.3. From $35.67 per donation in financial year 2009-2010, the values followed a decreasing trend (although in 2013-2014 there was a slight increase), concluding at $15.2 per donations 5 years later (i.e. a decrease of around 57.5%). Therefore, there exists an increase of more than 1073% of the number of donations in that period (see figure 5.4 - note the scale of y axis).

The Wikimedia Foundation annually creates finacial reports that help to mantain a transparent finantial system of the foundation. Annual plans and reports, forms and frequently asked questions (FAQ) can be found in [Foub]. Moreover, an independent auditors' report is published each financial year showing the balance sheet of the foundation (assets, liabilities and net assets) at the end of the last two financial years that are being compared. Figure 5.5 shows a comparison between the results at 30 June of 2015 and 2014 (i.e. end of financial years 2013-2014 and 2014-2015) presenting an increase in total support and revenue of a 44.47% (75,797,223 in 2015 whereas 52,465,287 in 2014) and a 14.58% in expenses (52,596,782 in 2015 whereas 45,900,745 in 2014). These results contributed to an increase of 45.53% in total net assets at the end of the year. This supports the argument that the business model used by Wikimedia Foundation is successful.

5.3 Data Collection

For this thesis, we collected the necessary performance and access data on a daily and monthly basis. In the remainder of this section, the different types of data gathered and their characteristics are presented.

5.3.1 Financial Data: Donations

In the evaluation of MediaWiki's profitability, financial data is essential. As explained in section 5.2, WikiMedia Foundation is the non- profit organization that supports financially a set of wikis (being Wikipedia the most popular). It is sustained through donations and data of these contributions since August 2008 (2008/08/03) to end of June 2016 (2016/06/30)[1] was collected (hereinafter dates are in format *yyyy/mm/dd* or *yyyy/mm* if no other specification is made). This data showed the total daily donations received at Wikimedia Foundation in USD, without considering refunds.

[1] https://frdata.wikimedia.org/

		2015	2014
Unrestricted net assets:			
Support and revenue:			
Donations and contributions	$	72,236,884	49,611,670
In-kind service revenue		235,570	370,602
Other income, net		270,296	650,618
Investment income, net		445,341	243,707
Release of restrictions on temporarily restricted net assets		2,609,132	1,588,690
Total support and revenue		75,797,223	52,465,287
Expenses:			
Salaries and wages		26,049,224	19,979,908
Awards and grants		4,522,689	5,704,791
Internet hosting		1,997,521	2,529,483
In-kind service expenses		235,570	370,602
Donations processing expenses		2,484,765	1,505,654
Professional service expenses		7,645,105	7,117,519
Other operating expenses		4,449,764	3,861,708
Travel and conferences		2,289,489	1,965,854
Depreciation and amortization		2,656,103	2,722,007
Special event expense, net		266,552	143,219
Total expenses		52,596,782	45,900,745
Temporarily restricted net assets:			
Contributions		5,014,816	2,971,086
Release of restrictions on temporarily restricted net assets		(2,609,132)	(1,588,690)
Increase in temporarily restricted net assets		2,405,684	1,382,396
Increase in net assets from operations		25,606,125	7,946,938
Foreign currency translation adjustment		(1,260,848)	338,959
Increase in net assets		24,345,277	8,285,897
Net assets at beginning of year		53,475,021	45,189,124
Net assets at end of year	$	77,820,298	53,475,021

Figure 5.5: Independent Auditors' Report 30 June 2015 and 2014[KPM]

5.3.2 Wikimedia Projects

User actions through Wikimedia projects are tracked and daily data of three areas was collected[2] for later analysis:

1. *Loading time*: 95 percentile of loading time of a search query results page (time between a user sends the request for a page and results are listed and provided for them by the servers, after the corresponding identification and ranking of probable related articles). Daily data from four sources was taken (i.e. desktop, mobile web and mobile app - Android or iOS-based) and also the mean of those four 95 percentile results was calculated.

2. *Clickthrough rate*: Data showing the percentage of visitors that had clicked on a link of Wikipedia Portal[3] to go to one of its projects.

3. *Search threshold passing rate*: Data showing the rate of users whose session dwell time at Wikimedia portal exceeds 10 seconds (i.e. time between a user arrives at the page and clicks into a link to another page or performs a search).

5.3.3 Wikipedia Statistics

Wikipedia, developed by Wikimedia Foundation in January 2001, aims to be a free set of encyclopedias in different languages. By the time this thesis was written, Wikipedia had almost 41 million articles in 283 languages (active). It also maintains some records about the statistics of this project[4]. Statistical information about all Wikipedia language versions can be found (i.e. active editors, edits per hour, number of articles, etc). However this data is only kept in a monthly basis and not daily records as needed so it could not be used and other sources were needed.

The English version, with 5,206,540 article count constitutes the biggest Wikipedia. Furthermore, around a 50% of the traffic to Wikimedia projects is addressed to the English edition of Wikipedia, constituting the most important one of the Wikimedia Foundation. Due to this, the analysis and study of this thesis was focused on this edition of the free online encyclopedia.

One collected metric is the number of unique devices that made one or more requests to the English Wikipedia project[5], both in web and mobile version, and also the total amount of unique devices was considered in the following analysis[6]. This study was focused on the English version of Wikipedia, as already mentioned, but available graphs support all the other projects too (e.g. Spanish Wikipedia, French Wikivoyage or the German Wiktionary). Data is collected on a daily basis but ranging between 2016/01/01 and 2016/06/30 due to the recent introduction of this metric measurement, based on *last access* cookie. Wikimedia Foundation

[2] https://datasets.wikimedia.org/aggregate-datasets
[3] www.wikipedia.org
[4] https://stats.wikimedia.org/EN/Sitemap.htm
[5] https://dumps.wikimedia.org/other/unique_devices/
[6] Graph available at https://analytics.wikimedia.org/

considered this metric as a good approximation of the total number of different users (closer than number of pageviews or registered users) accessing the project although they state it is an estimation as some people can use more than one device (laptop and mobile phone, for example) and other may share the same computer. But the number of user accounts (new created users at the English project of Wikipedia metric was also collected) is not linked to the total users as login is not a requirement for accessing the projects (with reading purposes). Wikimedia Foundation had been using comScore for this metric measurement but they decided to change the method due to the limitations. Desktop traffic measurements are the strength of comScore but due to the increasing mobile traffic, the results did not reflect accurately the desired metric.

Number of edits done by users and daily pageviews at English Wikipedia web were also collected from Wikimedia Foundation[7] [8], on a daily basis from 2015/07/01 to 2016/06/30. Values related to the number of created users at English Wikipedia were gathered as Wikipedia has public Wikipedia logs [9]. These values consider any user creation, without a limitation of contributions.

Daily data of some metrics related to search events at English Wikipedia was also collected from 2. When a user wants to perform a search query, he should: (1) start a session (search sessions), (2) wait until a results page is presented (result pages opened) and (3) click through to a listed article at the page in (2) (clickthroughs). Total of events (1), (2) and (3) was also calculated. These metrics were tracked from four sources: desktop, mobile web and mobile app (both Android and iOS-based). There exists also the option of obtaining zero results at the search (e.g. due to user spelling mistakes or no related entries). The rate of non-automata search queries at English Wikipedia that did not return any results is stored at the zero results rate metric.

5.4 Data Analysis

All the collected data was organized and analysed for the purpose of understanding the followed trends and establishing relations between the metrics.

5.4.1 Financial Data

The first dataset to be analysed was the donations received at Wikimedia Foundation (which, as already mentioned, supports financially some wiki projects such as Wikipedia or MediaWiki). Figure 5.6 shows the donations from 2009/01/01 to 2016/06/30 into four date basis: daily, weekly, monthly and yearly (from left to right).

As it can be seen in plot 5.6, daily, weekly and monthly donations follow a seasonal pattern with a rising trend, where some high peaks at the end of the years or beginning of the

[7] https://wikimediafoundation.org/wiki/Home
[8] Graphs available at https://analytics.wikimedia.org/
[9] https://en.wikipedia.org/wiki/Special:Log

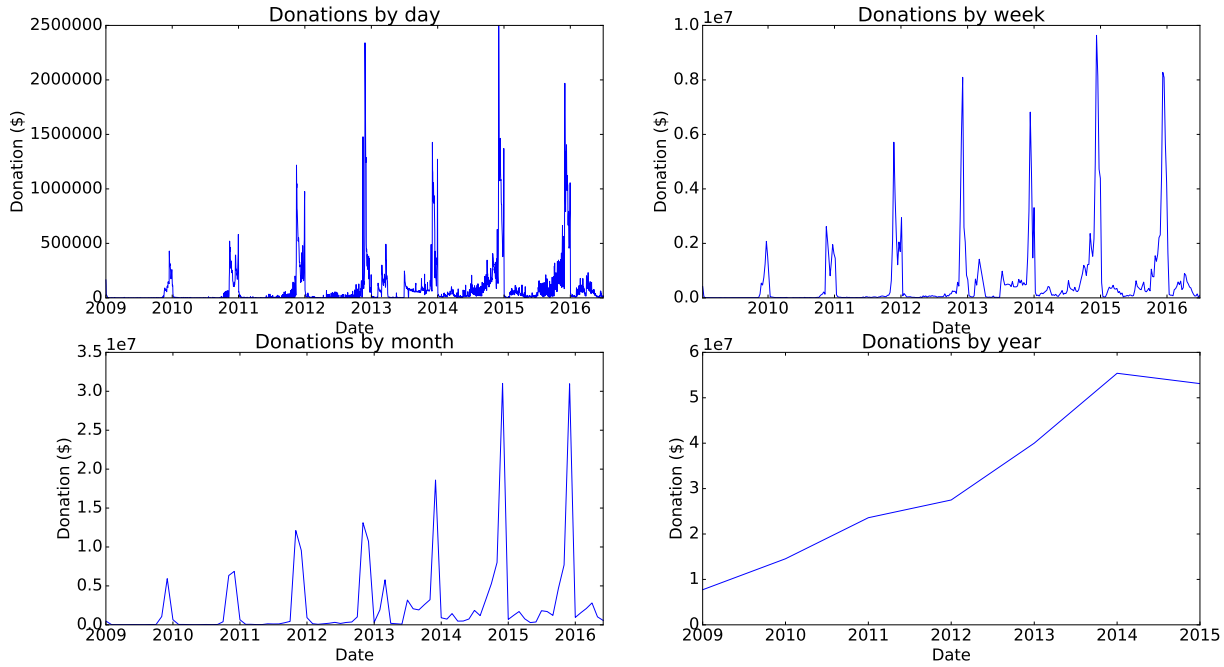

Figure 5.6: Wikimedia Foundation Donations

next one occurs. That is due to the English language fundraising campaign organized by Wikimedia Foundation. It takes place in late November and December and constitutes their largest fundraising campaign ($25M was the goal for the last campaign - conducted through December 2015 - finally reaching over $30M). Banners on the English Wikipedia together with emails to English-speaking foundation supporters compose the campaign.

Due to the repetition pattern found at the graphs, Dickey-Fuller test was performed on the daily basis and it was positive. Therefore, our data was stationary.

Furthermore, seasonal component was used to predict next year's period, based on the seasonal component, predicted trend through a linear model and residual results of previous year (plots are shown in figure 5.8) as the last two financial years have a similar behaviour. Seasonal decomposition results are shown in figure 5.7.

Due to the obtained results, the data analysis scope of this thesis was restricted to recent one year period from 1st July 2015 to 30th June 2016. Moreover, figure 5.8, behaviour of predicted year is similar to the one before and, therefore, it is used as the basis.

5.4.2 Wikimedia and Wikipedia Metrics

Data introduced at section 5.3 was arranged into the previously mentioned study period (2015/07/01 - 2016/06/30). The goal was analysing the strength of possible relationships

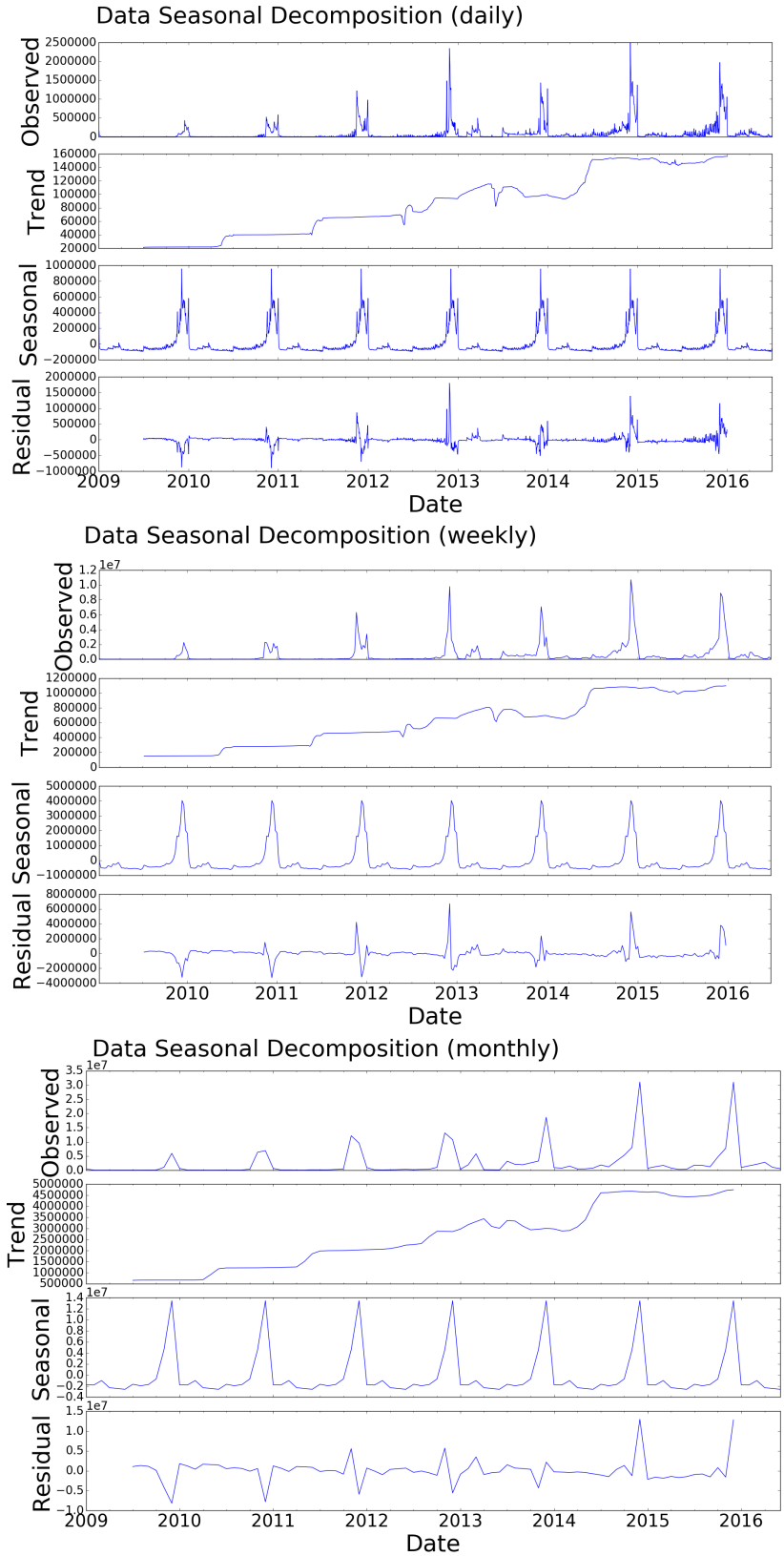

Figure 5.7: Seasonal decomposition

		TIME BASIS		
	METRIC	Day	Week	Month
Wikimedia projects	Desktop loading time (95 perc)	-0,119	-0,130	-0,222
	Mobile web loading time (95 perc)	0,170	0,280	0,403
	Android mobile app loading time (95 perc)	0,061	0,111	0,143
	iOS mobile app loading time (95 perc)	-0,035	0,005	-0,141
	Average loading time (95 perc)	0,004	0,073	-0,011
	Search threshold passing rate	0,192	0,228	0,236
	Clickthrough rate	-0,035	-0,101	-0,214
English Wikipedia	Pageviews	-0,060	0,021	-0,211
	Web unique device requests	0,130	0,328	0,393
	Mobile web unique device requests	-0,171	0,146	-0,288
	Total unique device requests	0,030	0,229	0,028
	New users	0,187	0,180	0,200
	Edits by users	-0,109	-0,406	-0,464
	Desktop search sessions events	0,014	0,063	-0,062
	Desktop results page events	-0,004	-0,010	-0,002
	Desktop clickthrough events	-0,080	-0,022	-0,218
	Total desktop events	-0,005	-0,012	-0,007
	Mobile search sessions events	-0,168	-0,056	-0,177
	Mobile results page events	-0,217	-0,163	-0,266
	Mobile clickthrough events	-0,233	-0,142	-0,237
	Total mobile events	-0,197	-0,104	-0,260
	Android mobile app search events	-0,171	-0,054	-0,113
	Android mobile app results page events	0,006	0,134	0,106
	Android mobile app clickthrough events	-0,161	-0,049	-0,089
	Android mobile app total events	-0,056	0,090	0,080
	iOS mobile app search events	-0,084	-0,087	-0,103
	iOS mobile app results page events	-0,085	-0,088	-0,102
	iOS mobile app clickthrough events	-0,084	-0,087	-0,102
	iOS mobile app total events	-0,085	-0,088	-0,102
	Zero results rate	0,200	0,147	-0,072

Table 5.1: Pearson's correlation coefficient between donations and metrics

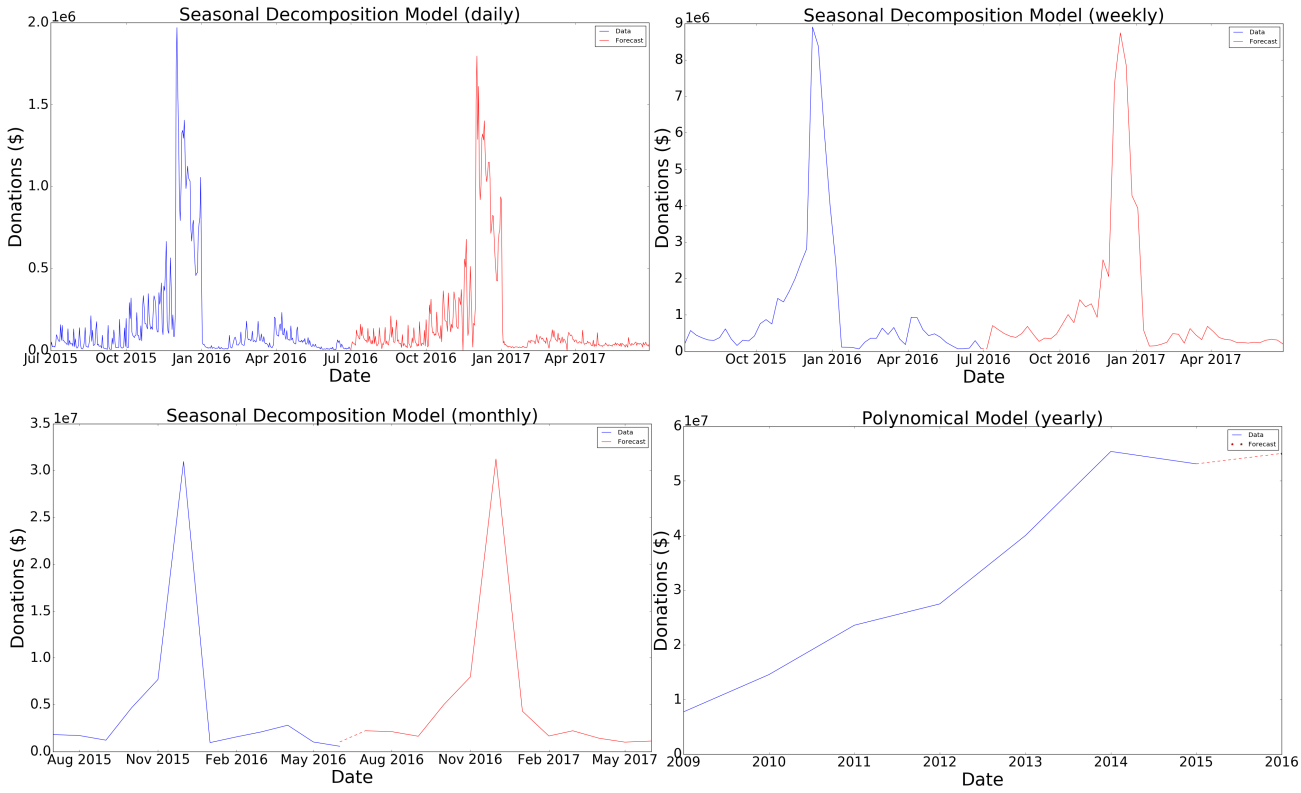

Figure 5.8: Wikimedia Foundation Donations prediction

between received donations at Wikimedia Foundation (i.e. revenue in this case) and the different studied metrics, in case they should be added to the utility definition for the study case. Followed procedure focused on Pearson's correlation coefficient (see section 2.3 for more information). Sets of data which were not complete for the study period took only into account the period were data existed. For later use, if needed, missing data in daily basis was replaced by the mean of existing values. Corresponding weekly and monthly values were calculated using that daily data. This analysis was performed in the already mentioned basis (daily, weekly and monthly) and the results are presented in table 5.1. Note that the obtained values (which represent the coefficients) are dimensionless, i.e. do not have units, as mentioned in section 2.3.

As it can be seen in the table, it is difficult to establish a linear relationship between the studied metrics and the received donations at the foundation and the three strongest are of *moderate* type: mobile web loading time (monthly, 0.403) and edits by users (weekly, -0.406; monthly, -0.464). The first one suggest that mobile web users are not affected strongly by the delay time, as their speed expectations are not as high as, for example, desktop users. The second one suggest that there exists a slightly higher relation between number of users editions at English Wikipedia and donations received as it appeared both in weekly and monthly basis. These two coefficients are negative, meaning it is an inverse moderate relationship and therefore, the more editions, the less donations received and vice versa. Finding mistakes and correcting them seems to slightly influence users to reduce their economic contributions.

In general, correlation coefficient becomes stronger if the studied time basis is longer. This could be expected as the cumulative influence of Wikimedia performance to donations is better found in a longer-term basis (e.g. users donate some days after their experience).

The conclusion of these performance metrics analysis is that there is very difficult to establish a linear association between them and an increase or reduction of received donations at Wikim. Foundation. Only the number of user editions at English Wikipedia scored a - negative - moderate connection and, as it appeared both in weekly and monthly basis, it is considered afterwards at the utility analysis.

5.5 Derived Monthly Revenue Model

The definition of utility function specified in this document considers two main terms: $total_{revenue}$ and $cost$, both with some subterms. This section develops one of the terms associated to the revenue: rpt_{basis}. As this application does not use a sales revenue model, transactions are associated in this case to number of pageviews. Considering real data from Wikimedia Foundation and English Wikipedia (see sections 5.3 and 5.4), some models for the first semester of the year were defined. This period was chosen because, although studied data ranged for a year (2015/07/01 - 2016/06/30), metrics used to collect data regarding the number of users changed on January 2016.

5.5.1 Monthly Models

Models were proposed for the first 6 months of the year, studying the *revenue per transaction* in a daily and weekly basis. Days range from 1 to 30, 31 or 29 depending on the considered month and weeks are counted in an ordered way. The value for the first week of the month is 1, for the second week 2, and so on. Moreover, a simplified and general fragment represent the followed implementation for simplicity and after that, results are presented and are used afterwards at the evaluation.

First of all, needed modules are imported for later use and studied month period is defined, as well as indicating the data files.

```
1  import pandas as pd
2  import matplotlib.pylab as plt
3  import numpy as np
4  from scipy.optimize import curve_fit
5  from matplotlib import gridspec
6
7  FIRST_DAY = '01-01-2016' #mm-dd-yyyy format is followed
8  LAST_DAY = '01-31-2016'
9  DATE_INDEX_COLUMN = 'Date'
10
11 FILES = ['csv-file-with-daily-data', 'csv-file-with-weekly-data']
12 TITLES = ['Daily basis', 'Weekly basis']
```

After that, the model function has to be defined. This depends on the data shape and the desire of the user as any function allowed by python or one of its modules can be used. In this case, an exponential and a polynomial function are shown, with three and four coefficients respectively but users can add any number of possible coefficients to be calculated provided that its number does not exceed the number of data points available.

```
def exponential(x, a, b, c):
    return a*np.exp(-b*x)+c
def polynomial(x, a, b, c, d):
    return a*np.power(x,3)+b*np.power(x,2)c*x+d
```

A dateparser is used while reading the csv files so that python is able to manage the dates following the indicated format (in this case *month/day/year*) used at the file.

```
dateparser = lambda dates: pd.datetime.strptime(dates, '%m/%d/%Y')
data = pd.read_csv(desired_file, parse_dates=True, date_parser=dateparser,
                                        index_col=DATE_INDEX_COLUMN)
```

After filtering the data with the desired values at the defined period and configuring the plots, the *curve_fit* algorithm was applied (see section 2.5.2 for more information). X points for the plot are stated as *x_data*. The initial point where the algorithm starts its computation is noted as *initial-point* (although it is an optional parameter) and *amount-of-desired-points* will define the fitting function. Note that *p0* must have the same number of components as coefficients of previously defined modelling function. Finally, *model-function* has to be substituted by the name of the modelling function used. In this case it can be replaced by *exponential* or *polynomial*.

```
    popt, pcov = curve_fit(model-function, x_data, working_data, p0=initial-
                                        point)
    x_fitting_points = np.linspace(1, working_data.size, amount-of-desired-
                                        points)
    fitting_function = exponential(x_fitting_points, *popt)
```

Where *popt* contains the calculated coefficients minimizing the least square error for the modelling function if a solution was found. Then, a plot showing the calculated function was presented.

Obtained results for all the studied months are introduced, with 2 plots per month. The left one is based on daily analysis and the right one on weekly data. They can be seen at figures 5.9, 5.10, 5.11, 5.12, 5.13 and 5.14. Resulting functions were:

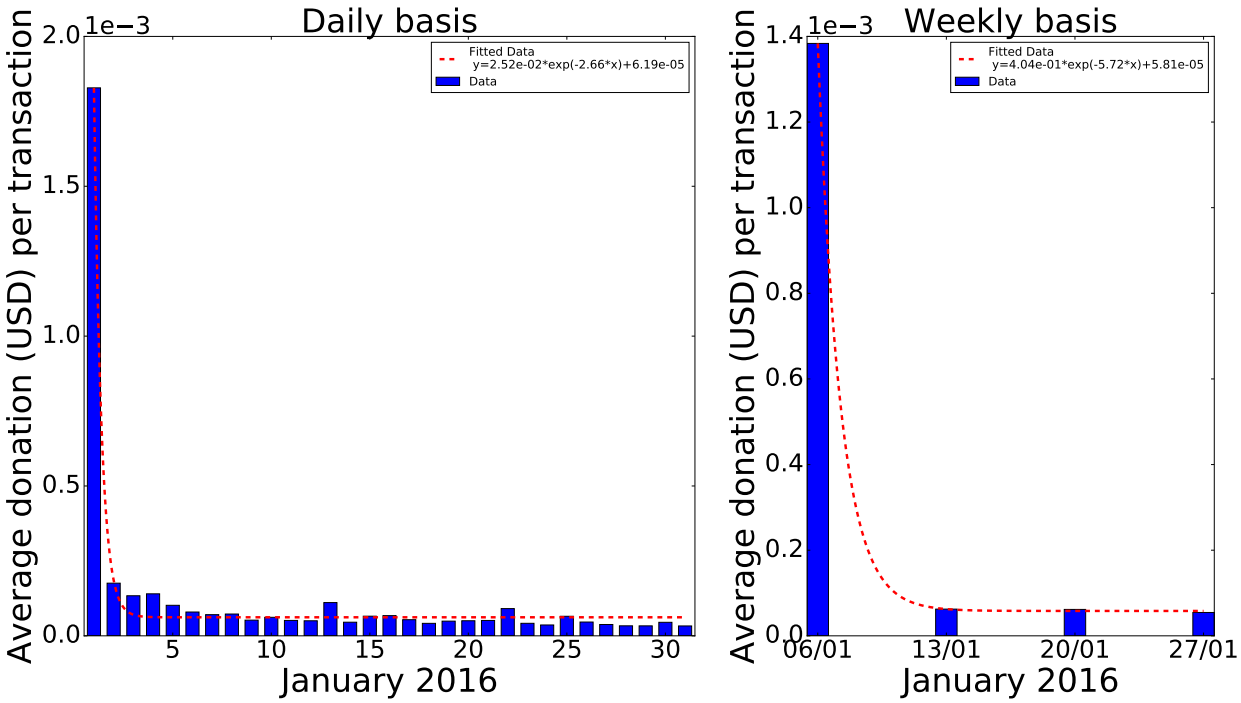

Figure 5.9: January Model

January

Figure 5.9 shows the modelled functions for this month:

$$\begin{cases} rpt_{day} = 2.52 * 10^{-2} * e^{-2.66x} + 6.19 * 10^{-5} \\ rpt_{week} = 4.04 * 10^{-1} * e^{-5.72x} + 5.81 * 10^{-5} \end{cases}$$

February

Figure 5.10 shows the modelled functions for this month:

$$\begin{cases} rpt_{day} = 1.02 * 10^{-5}x + 8.05 * 10^{-5} * e^{3.04*10^{-2}x} * sin(8.98 * 10^{-1}x + 29.6) + 6.11 * 10^{-5} \\ rpt_{week} = -2.62 * 10^{-5}x^2 + 1.85 * 10^{-4}x - 1.24 * 10^{-4} \end{cases}$$

March

Figure 5.11 shows the modelled functions for this month:

$$\begin{cases} rpt_{day} = 1.89 * 10^{-8}x^4 - 10^{-6}x^3 + 1.58 * 10^{-5}x^2 - 7.69 * 10^{-5}x + 3.58 * 10^{-4} \\ rpt_{week} = 3.74 * 10^{-5}x^4 - 4.59 * 10^{-4}x^3 + 1.93 * 10^{-3}x^2 - 3.23 * 10^{-3}x + 2.07 * 10^{-3} \end{cases}$$

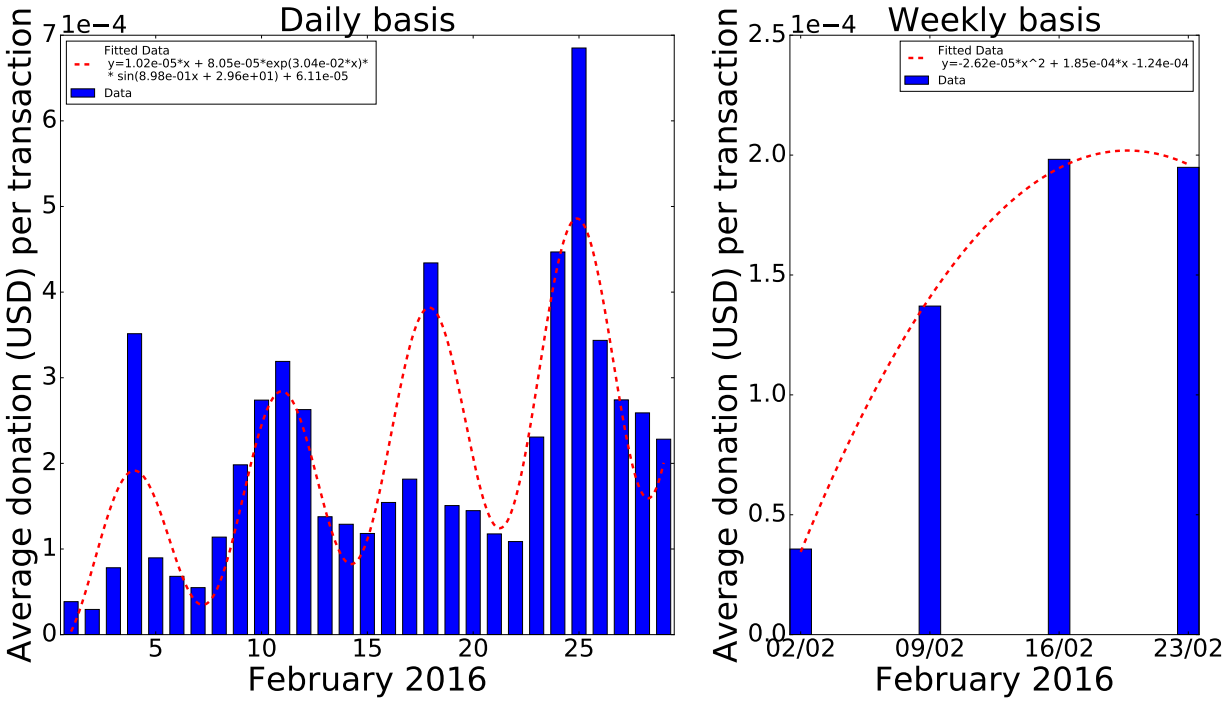

Figure 5.10: February Model

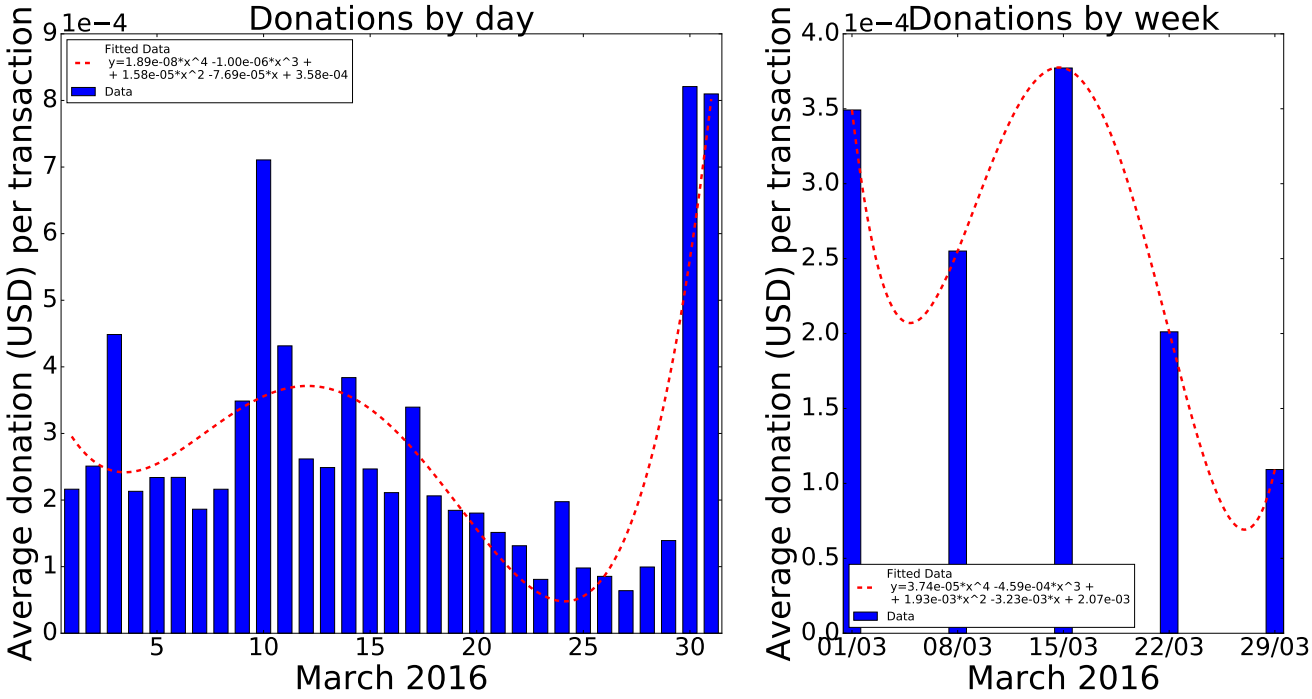

Figure 5.11: March Model

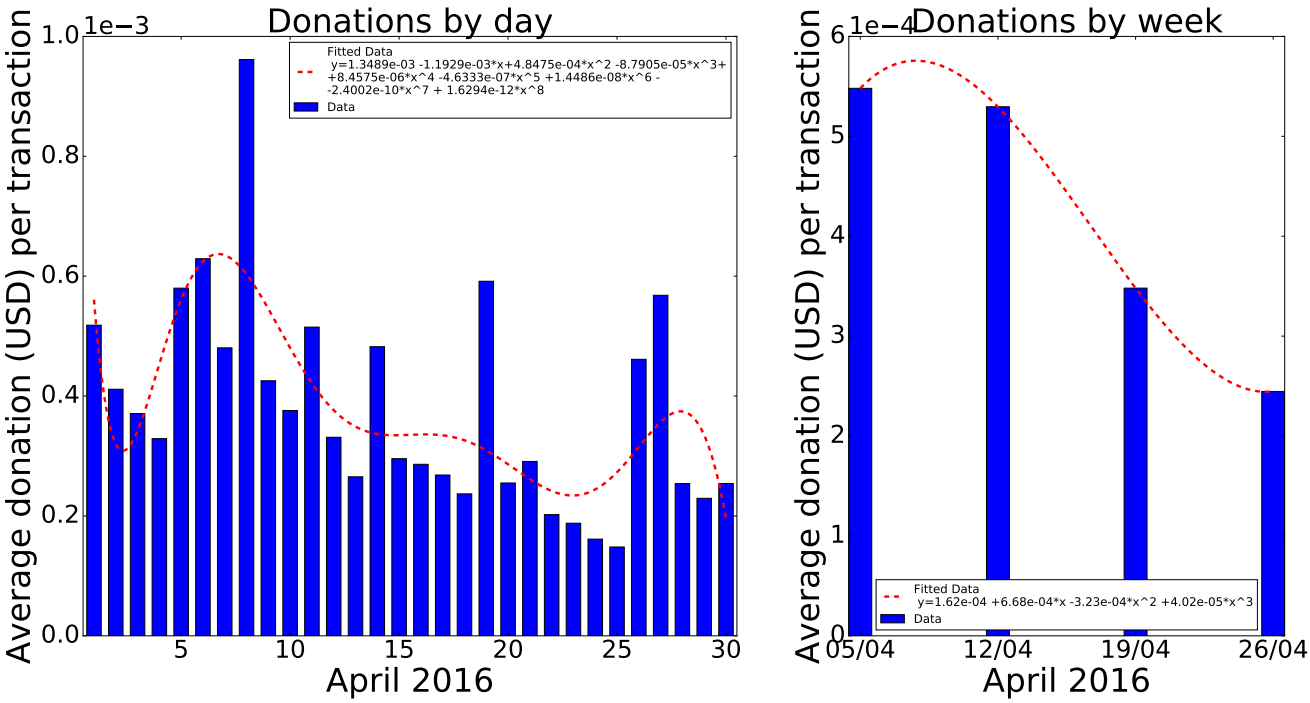

Figure 5.12: April Model

April

Figure 5.12 shows the modelled functions for this month:

$$\begin{cases} rpt_{day} = 1.6294 * 10^{-12}x^8 - 2.4002 * 10^{-10}x^7 + 1.4486 * 10^{-8}x^6 - 4.6333 * 10^{-7}x^5 + 8.4574* \\ \qquad * 10^{-6}x^4 - 8.7905 * 10^{-5}x^3 + 4.8475 * 10^{-4}x^2 - 1.1929 * 10^{-3}x + 1.3489 * 10^{-3} \\ rpt_{week} = 4.02 * 10^{-5}x^3 - 3.23 * 10^{-4}x^2 + 6.68 * 10^{-4}x + 1.62 * 10^{-4} \end{cases}$$

May

Figure 5.13 shows the modelled functions for this month:

$$\begin{cases} rpt_{day} = 2.03 * 10^{-8}x^3 - 9.93 * 10^{-7}x^2 + 5.4 * 10^{-6}x + 2.2 * 10^{-4} \\ rpt_{week} = 3.29 * 10^{-6}x^3 - 2.67 * 10^{-5}x^2 - 7.56 * 10^{-7}x + 3.05 * 10^{-4} \end{cases}$$

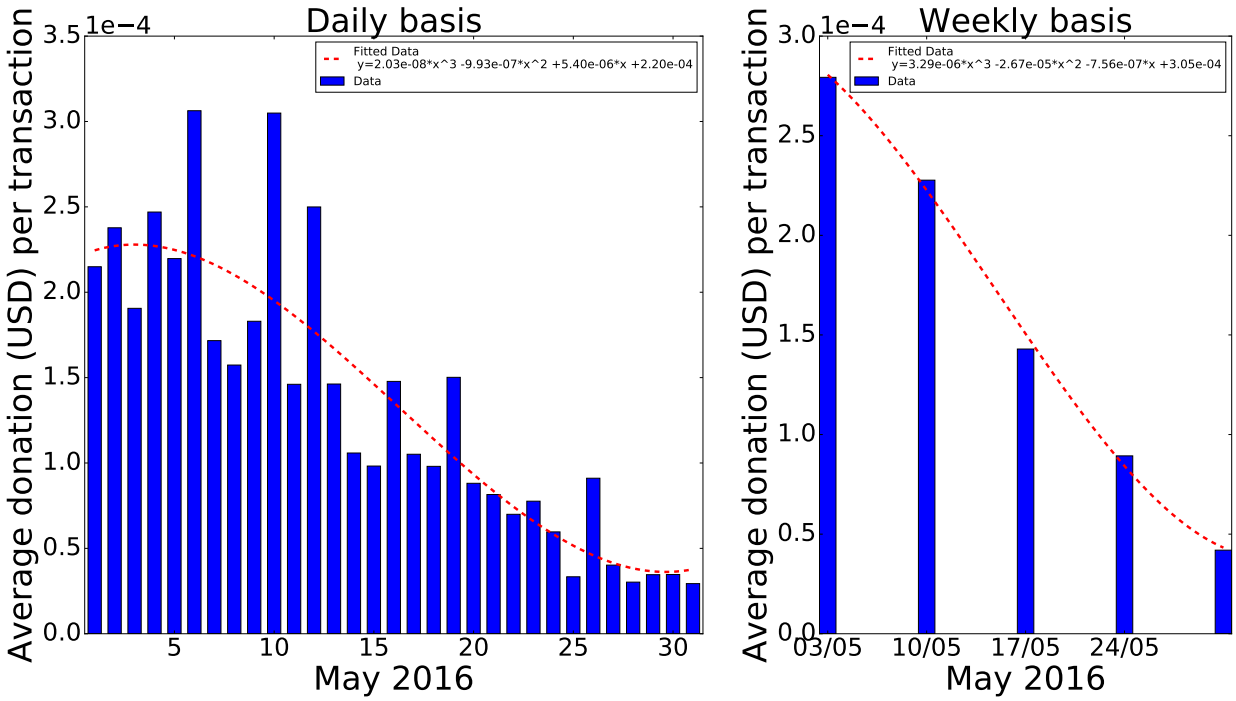

Figure 5.13: May Model

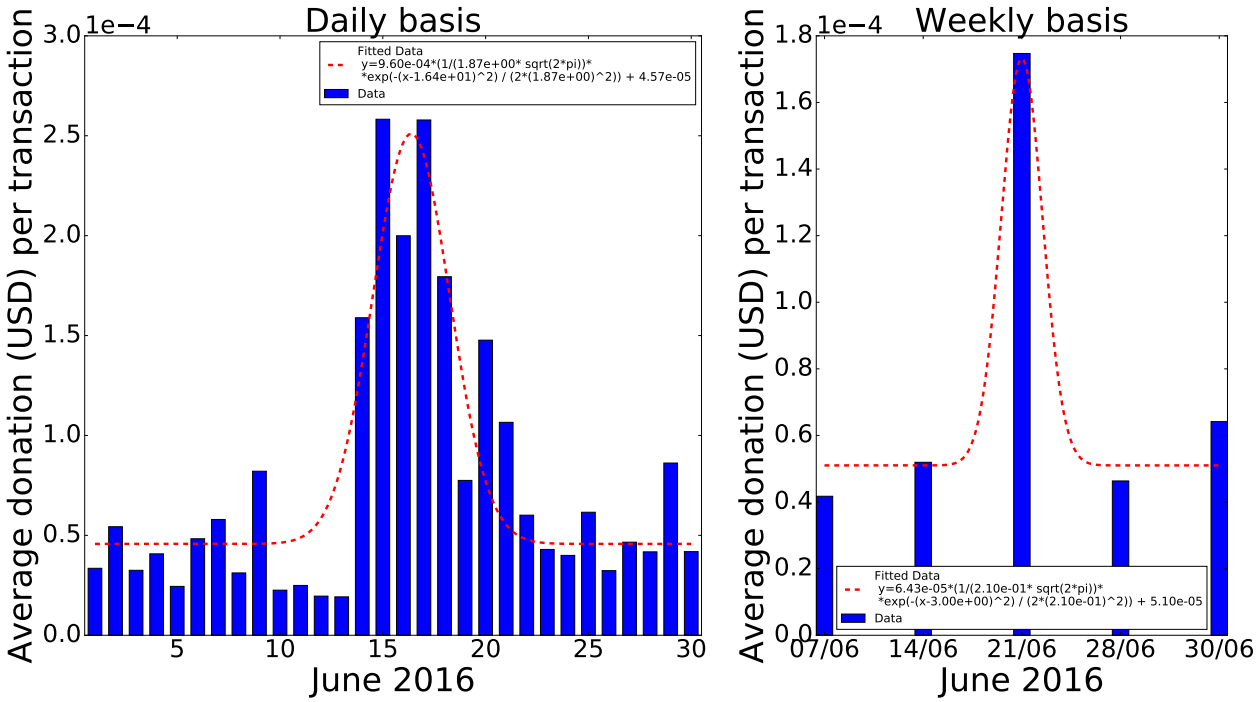

Figure 5.14: June Model

June

Figure 5.14 shows the modelled functions for this month:

$$\begin{cases} rpt_{day} = 9.6 * 10^{-4} * \left(\frac{1}{1.87*\sqrt{2\pi}}\right) * e^{-\frac{(x-16.4)^2}{2*(1.87)^2}} + 4.57 * 10^{-5} \\ rpt_{week} = 6.43 * 10^{-5} * \left(\frac{1}{2.1*10^{-1}*\sqrt{2\pi}}\right) * e^{-\frac{(x-3)^2}{2*(2.1*10^{-1})^2}} + 5.1 * 10^{-5} \end{cases}$$

5.6 Discussion

Chapter 5 summarizes the analysis of Wikipedia's revenue model. As explained here, it is not based on any up-to-date models explained in *Chapter 4* but on users' and benefactors' donations to Wikimedia Foundation. Studying the published financial information shows that over the last six years there is a rising trend at received funds whereas the average amount of money per donation decreases.

After that, the topology of MediaWiki (software addressed at the beginning to cover Wikipedia needs) is reviewed as studying the possible cloud deployments on a components level gives more flexibility than considering the application as one compact element.

Finally, the revenue model is derived. For that purpose, first the collected data is presented and then analysed to find possible direct relationships between them and the received donations. Finally, the monthly models for the first semester (particular for each month) are calculated and, as well as the metric analysis results, are used at the utility based evaluation, whose definition is done at the following chapter.

6 Utility Functions

As already explained through this document, utility functions are used as the basis for the comparison of viable application distributions. This chapter develops and explains the followed utility-based analysis based on [Fre16] (see section 2.4 for further information) with some modifications.

6.1 Concept

The presence of a set X of elements that are aimed to be compared, makes necessary the definition of a *utility function* u, that assigns a number in $\mathbb{R}$ domain to each object $x \in X$ ($u : X \rightarrow \mathbb{R}$). In our case, as already mentioned, function u was defined as a trade-off between revenue and cost. Optimizing an application's distribution is done by evaluating the utility functions with a set of parameters called *application profile* (P). Nonetheless, utility functions are not unique. Depending on the dimension under study or the metrics considered, different ones can be defined (e.g. topology chosen will have an influence on the evaluated cost).

Before defining the utility function, some elements are needed:

- A set of topologies Λ for a web application can be generated, included viable inferred topologies $T^{\mu}_{(i)}$, $i \geq 1$ where $T^{\mu}_{(i)} \in \Lambda$. Set $T^{\mu}_{(i)}$ is composed by two subsets: $T_{(i)} = \{T^{\alpha}_{(i)}, T^{\gamma}_{(i)}\}$, $i \geq 1$, representing the associated α and γ topologies of the distribution.
- A set of requirements or constraints Γ (R in [Fre16])are also defined: $\Gamma = \{r_1, r_2, ..., r_n\}$. For example, possibility of scalability for future applications.
- A set of ordered time intervals Φ, as the basis for the application topology analysis. $\Phi = \{t_1, t_2, .., t_q\}$, where each t_q is determined by two date-time values, $t_q^{min} and t_q^{max}$, that define the beginning and the end of the interval $t_q = \{t_q^{min}, t_q^{max}\}$.

  For example, in a simple daily basis case defined by a pair of time intervals $\Phi = \{t_1, t_2\}$:

  $t_1^{min} = \{2016/02/12 \quad 00:00:00\}$ $t_1^{max} = \{2016/02/12 \quad 23:59:59\}$

  $t_2 = \{\{2016/02/13 \quad 00:00:00\}, \{2016/02/13 \quad 23:59:59\}\}$

Afterwards, the utility function can be completely defined:

$$u(T^{\mu}, \Gamma, t_q, P) = total\_revenue(T^{\mu}, t_q, P) - cost(T^{\mu}, \Gamma, t_q, P) \tag{6.1}$$

In our case, T^{μ}, Γ, t_q, P define object X, so we can rewrite the expression as $u : T^{\mu}, \Gamma, t_q, P \rightarrow \mathbb{R}$. Then, u is composed by $total\_revenue : T^{\mu}, t_q, P \rightarrow \mathbb{R}$, which takes into account the expected

revenue at a concrete time interval and some QoS metrics, and $cost : T^{\mu}, \Gamma, t_q, P \rightarrow \mathbb{R}$, calculating the predicted cost of the distribution for chosen topology. These elements are developed in following sections 6.2 and 6.3 respectively.

6.2 Revenue

This section develops the *total_revenue* function, defined at 6.1. Three terms have an influence on it: rev_{exp}, $satisfaction$ and $availab$.

- $rev_{exp} : T^{\mu}, t_q, P \rightarrow \mathbb{R}$, expected revenue of the application during time interval $t_q \in \Phi$.
- $satisfaction : t_q \rightarrow [0, 1]$, ratio of user satisfaction for time interval $t_q \in \Phi$, which has a strong influence on the total revenue of the application.
- $availab : T^{\mu}, \Gamma, t_q, P \rightarrow [0, 1]$, system availability will also impact on possible revenue as it is an important QoS metric.

By expressing the concept with an equation:

$$total\_revenue(T^{\mu}, t_q, P) = rev_{exp}(T^{\mu}, t_q, P) * satisfaction(t_q) * availab(T^{\mu}, \Gamma, t_q, P)$$

Expected Revenue

The expected revenue measures the predicted economical profits obtained by the application at a time interval t_q. It can be calculated as the sum of individual expected revenues over all the time intervals Φ

$$rev_{exp\_basis}(T^{\mu}, W, t, P) = \sum_{t=t_1}^{t_q} tpu(t) * rpt(t) * \#users(t)$$

where:

- $tpu(t)$: shows the average number of transactions per user at time period $t \in \Phi$.
- $rpt_{basis}(t)$: shows the average revenue per transaction at time period $t \in \Phi$. A model is derived considering the defined data *basis* under consideration (e.g. daily, weekly, monthly).
- $\#users(t)$: average number of users at time period $t \in \Phi$.

Satisfaction

User satisfaction is taken into account when considering the total possible revenue of the application. The performance data analysis exposed at section 5.4.2 shows a moderate relationship between a reduction of donations and the number of editions at English Wikipedia. Consequently, as already mentioned, finding errors is considered as an indicator of users' dissatisfaction and is contemplated in this term. The chosen definition in this document was:

$$satisfaction = 1 - \left[\frac{average\_load\_time}{max\_acc\_load\_time} * w_{lt} + zero\_rate * w_z + \left(\frac{user\_edits}{total\_articles} \right) * w_e \right]$$

where:

average_load_time denotes the average user perceived loading time at a time interval $t_q \in \Phi$,

max_acc_load_time is the maximum accepted loading time for users,

zero_rate represents zero rate ratio obtained by users' search query,

user_edits considers the number of editions performed by users,

total_articles is the number of articles contained at the considered version,

w_{lt} represents the dissatisfaction weight associated to the loading time,

w_z denotes dissatisfaction weight associated to the zero results rate and

w_e is the dissatisfaction weight associated to edits.

[Nah04] studies the tolerable waiting time for web users, also mentioning and comparing previous researches. It mentions that the acceptable time varies depending on the type of page, as user expectation regarding simple content pages are lower than pages with more complex content (e.g. available graphics or multimedia - such as videos - at the page). The interval between 2 and 4 seconds represents the time range when a change in users behaviour takes place and other mentioned studies also remark 2 seconds as the acceptable delay. Therefore, for this analysis 2 seconds are assumed as the minimum tolerable average loading time, without having influence on user satisfaction. [Nah04] concludes also that 15 seconds is the maximum tolerable time. Therefore, $max\_acc\_load\_time = 15sec$. Furthermore, weights related to loading time, zero results rate and edits (w_{lt}, w_z and w_e respectively) should be defined. In this document, the loading time and the zero results rate are assumed to have the same influence and, as the relationship found between donations and editions was moderate and not strong (according to Pearson's coefficient interpretation table 2.1), its weight is considered as half of the other two. Therefore, it was defined $w_{lt} = w_z = 2/5$ and $w_e = 1/5$.

Availability

Service availability has an impact on users engagement, it is also considered as a *total_revenue* term. Consequently, for topology $T^{\mu}_{(i)}$, consisting of N instances or elements (e.g. physical server) considered independent, average global availability can be calculated as a standard average:

$$availab(T^{\mu}, \Gamma, t_q, P) = \frac{1}{N} * \sum_{j=1}^{N} availability(j) \tag{6.2}$$

For example, suppose that an application topology distribution T^{μ}_1 entails having a physical server and a cloud instance. The first element fails statistically two hours per three months and chosen cloud provider ensures a minimum availability of 99.85%. Following equation 6.2, first step is calculating the individual availabilities:

- Cloud instance availability is direct, as provider already stated it.
- Physical server availability should be derived:

$$\begin{aligned} 1\,natural\,year &= 8760\,hours \\ \frac{2\,hours\,failure}{3\,months} * \frac{12\,months}{year} &= \frac{8\,hours\,failure}{year} \\ availability_{physical\_server} &= \frac{8760 - 8}{8760} \\ &\approx 0.999087 \\ &\approx 99.91\% \end{aligned}$$

And then, total availability of topology T^{μ}_1 can be calculated with 6.2:

$$\begin{aligned} availab(T^{\mu}_1, \Gamma, t_q, P) &= \frac{1}{2} * \sum_{j=1}^{2} availability(j) \\ &\approx \frac{0.9991 + 0.9985}{2} \\ &\approx 0.99879 \\ &\approx 99.88\% \end{aligned}$$

6.3 Cost

This section develops the *cost* function, defined at 6.1. As [Fre16] defined, the associated cost for a distribution of an application on the cloud implies the aggregation of a fixed

cost, associated to the cloud offerings and the adaptation costs to satisfy a set of defined requirements: $cost(T^{\mu}, \Gamma, t_q, P) = cost_{fixed}(T^{\mu}, \Gamma, t_q) + cost_{adaptation}(T^{\mu}, \Gamma, t_q, P)$

$cost_{fixed} : T^{\mu}, \Gamma, t_q \rightarrow \mathbb{R}_+$ measures the direct economical charges for provisioning all the cloud instances or elements needed for the deployment to develop topology T_i^{μ} fulfilling requirements Γ at time interval t_q.

$cost_{adaptation} : T^{\mu}, \Gamma, t_q, P \rightarrow \mathbb{R}_+$ measures the possible needed adaptations for topology T_i^{μ} to satisfy all requirements in Γ.

$$cost_{fixed} = \sum cost_{cloud\_offerings}$$

For example, if the viable application topology T_1^{μ} implies the need of AWS instances *t2.small*, *m4.large* and hosts the Web Shop front end on a physical IBM zSeries server, the $cost_{fixed}$ would be calculated as:

$$cost_{fixed} = \sum cost_{cloud\_elements} =$$
$$= cost_{t2.small} + cost_{m4.large} + cost_{zSeries}$$

where *cost* of cloud instances would be fixed by the provider and $cost_{zSeries}$ could be simplified and calculated as the aggregation of the costs associated to the N_{CPU} CPU cores at the server cluster working Y years[SAH+16]:

$$cost_{on-premise} = \frac{\left(1 - \frac{1}{\sqrt{2}}\right) * \sum_{t=0}^{Y-1} \frac{C_t}{(1+k)^t}}{\left[1 - \left(\frac{1}{\sqrt{2}}\right)^Y\right] * N_{CPU} * H * \mu}$$

C_0 is the acquisition cost and

$C_{1..Y}$ are the annual maintenance costs over the working years,

k represents the investment,

h denotes the operational hours expectation,

μ is the expected utilization of the cluster

Fixed cost is also determined by the application profile under study. For example, AWS instances can have different charges if it is on demand or reserved, and the chosen pay method. *EC2 t2.small* is used to ilustrate this[1]. Also no upfront and partial upfront cases are supported but not considered here for simplicity.

$$cost_{EC2\_t2.small} = \begin{cases} 0.028 * h & \textit{if instance = on-demand} \\ 163 + 0.019 * h & \textit{reserved 1 year, all upfront} \\ 332 + 0.013 * h & \textit{reserved 3 years, all upfront} \end{cases}$$

[1]Prices of September 2016 at https://aws.amazon.com/ec2/pricing/

where h represents the hours of instance usage.

On the other hand, $cost_{adaptation}$ represents the aggregation of additional costs (e.g. scaling needs due to a peak at workload or the requirement of having a database replica) for topology $T^{\mu}_{(i)}$ at a concrete time interval t_q, ensuring that all the requirements Γ are satisfied with the application profile values P.

7 Case Study: MediaWiki and English Wikipedia

7.1 Evaluation

In this chapter, a practical case of the utility-based analysis explained before is carried out. MediaWiki case is studied with the real data mentioned in sections 5.3 and 5.4. Chapter 6 develops the utility function concept and definition though this thesis as the trade-off between the total revenue and the cost (see equation 6.1). Both terms are developed separately and afterwards the utility is calculated. In the evaluation, *daily* and *monthly* basis are chosen following the modelling *rpt* equations basis.

Therefore, defining an application profile (P) is necessary for our purpose: Our application will run from January to June, both inclusively, with a 24/7 system, implying 4344 hours in total. Moreover, West Europe is assumed as the desired infrastructure location and only one provider is requested for all the instances to avoid inter-provider latencies. In this document, four of the most popular cloud providers are considered: Amazon Web Services (AWS), Windows Azure, Rackspace and Google Compute Engine (GCE).

Table 7.1 shows a subset of viable topologies for MediaWiki application which would be under study. Furthermore, some acyclic graphs of topologies of table 7.1 are depicted in figure 7.1. For simplicity, it only includes the cases considering AWS as the provider.

7.1.1 Revenue

In this document, *total_revenue* term is related to the $T^{\mu}_{(i)}$, $1 \leq i \leq 13$ where $T^{\mu}_{(i)} \in \Lambda$ with already stated equation:

$$total\_revenue(T^{\mu}, t_q, P) = rev_{exp}(T^{\mu}, t_q, P) * satisfaction(t_q) * availab(T^{\mu}, \Gamma, t_q, P)$$

Expected revenue and satisfaction are calculated depending on the time basis and the month under study but availability according to chosen providers is constant for $\forall t_q \in \Phi$. According to AWS, its service-level agreement offers an uptime percentage of at least 99.95% per month for EC2[1] and RDS[2], as well as Google Compute Engine does[3]. Azure also guarantees an availability of 99.95% for VM[4]. Finally, Rackspace offers an availability of at least 99.9% for all their cloud instances[5].

[1] As of June 1, 2013 - https://aws.amazon.com/es/ec2/sla/
[2] As of March 25, 2016 - https://aws.amazon.com/es/rds/sla/
[3] As of August 20, 2016 - https://cloud.google.com/compute/sla
[4] As of July, 2016 - https://azure.microsoft.com/en-us/support/legal/sla/virtual-machines/v1_2/
[5] As of February 8, 2015 - https://www.rackspace.com/information/legal/cloud/sla/

Topology	Provider	MediaWiki Front-End	MediaWiki Back-End	Region	Cost on demand
T_1^μ	AWS	EC2 t2.large (Linux)	EC2 m3.xlarge (Linux)	EU (IR)	0.405 ($/h)
T_2^μ	AWS	EC2 m4.2xlarge (Linux)		EU (IR)	0.528 ($/h)
T_3^μ	AWS	EC2 m4.large (Linux)	EC2 r3.large (Linux)	EU (IR)	0.317 ($/h)
T_4^μ	AWS	EC2 m4.large (Linux)	RDS db.r3.large (max stor.)	EU (IR)	1.463 ($/h)
T_5^μ	Azure	DS2 (Linux)	DS3 (Linux)	West Europe	0.6128 ($/h)
T_6^μ	Azure	DS4 (Linux)		West Europe	0.8171 ($/h)
T_7^μ	Azure	DS2 (Linux)	DS11 (Linux)	West Europe	0.4767 ($/h)
T_8^μ	Rackspace	Cloud Server 8GB (Linux)	Cloud Server 8GB(x2) (Linux)	GB	0.768 ($/h)
T_9^μ	Rackspace	Cloud Server 8GB (Linux)	Cloud Server (Memory) 15GB (Linux)	GB	0.436 ($/h)
T_{10}^μ	Rackspace	Cloud Server 8GB (Linux)	Cloud DB 16	GB	1.216 ($/h)
T_{11}^μ	GCE	n1-standard-2 (Linux)	n1-standard-4 (Linux)	Europe	0.234 ($/h)
T_{12}^μ	GCE	n1-standard-8 (Linux)		Europe	0.312 ($/h)
T_{13}^μ	GCE	n1-standard-2 (Linux)	n1-highmem-2	Europe	0.175 ($/h)

Table 7.1: Subset of Viable Topologies for MediaWiki Application

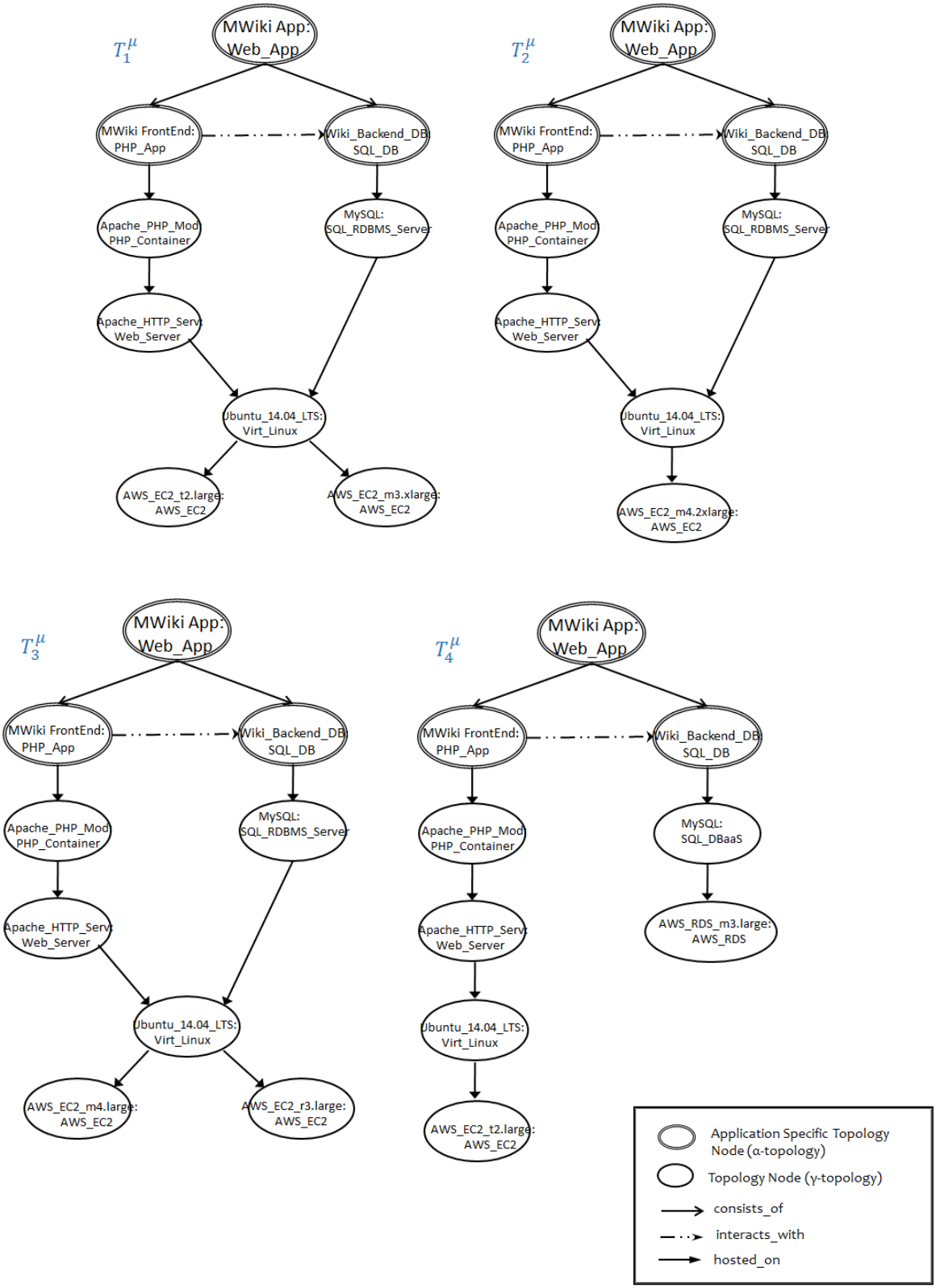

Figure 7.1: Viable Topologies of MediaWiki Application

Therefore, the total availability for the studied subset of viable topologies can be calculated:

$$availability = \begin{cases} 99.95\% & for \quad T_1^\mu \\ 99.95\% & for \quad T_2^\mu \\ 99.95\% & for \quad T_3^\mu \\ 99.95\% & for \quad T_4^\mu \\ 99.95\% & for \quad T_5^\mu \\ 99.95\% & for \quad T_6^\mu \\ 99.95\% & for \quad T_7^\mu \\ 99.9\% & for \quad T_8^\mu \\ 99.9\% & for \quad T_9^\mu \\ 99.9\% & for \quad T_{10}^\mu \\ 99.95\% & for \quad T_{11}^\mu \\ 99.95\% & for \quad T_{12}^\mu \\ 99.95\% & for \quad T_{13}^\mu \end{cases}$$

Moreover, user satisfaction was calculated using equation developed at section 6.2 and the corresponding collected data, keeping the assumption that a loading time less than 2 seconds does not have an influence on users satisfaction, and the weights consideration defined at section 6.2 (see figure 7.2):

$$satisfaction = 1 - \left[\frac{average\_load\_time}{max\_acc\_load\_time} * 0.4 + zero\_rate * 0.4 + \left(\frac{user\_edits}{total\_articles} \right) * 0.2 \right]$$

On the other hand, expected revenue should be calculated with the modelling formulae obtained in section 6. A non-leap year was considered, with the same week distributions as from 2016/01/01 to 2016/06/30. As the rpt are calculated for each month, the calculations are performed for a concrete month and are summed afterwards. Therefore,

$$(rev_{exp} * satisf)_{basis} = \sum_{mnth=Jan}^{June} \sum_{t=t_1}^{t_q} tpu(mnth,t) * rpt(mnth,t) * \#users(mnth,t) * satisf(mnth,t)$$

In this case tpu term (transactions per user) considers the number of pageviews as transactions as mentioned in section 5.5. This document does not calculate its value on the studied cloud services but considers the values obtained at the collected statistical data (section 5.3) as the basis for the utility calculation. Term tpu for this period is depicted in figure 7.3 whereas $\#users$ is represented at figure 7.4. Both graphs show daily and monthly records.

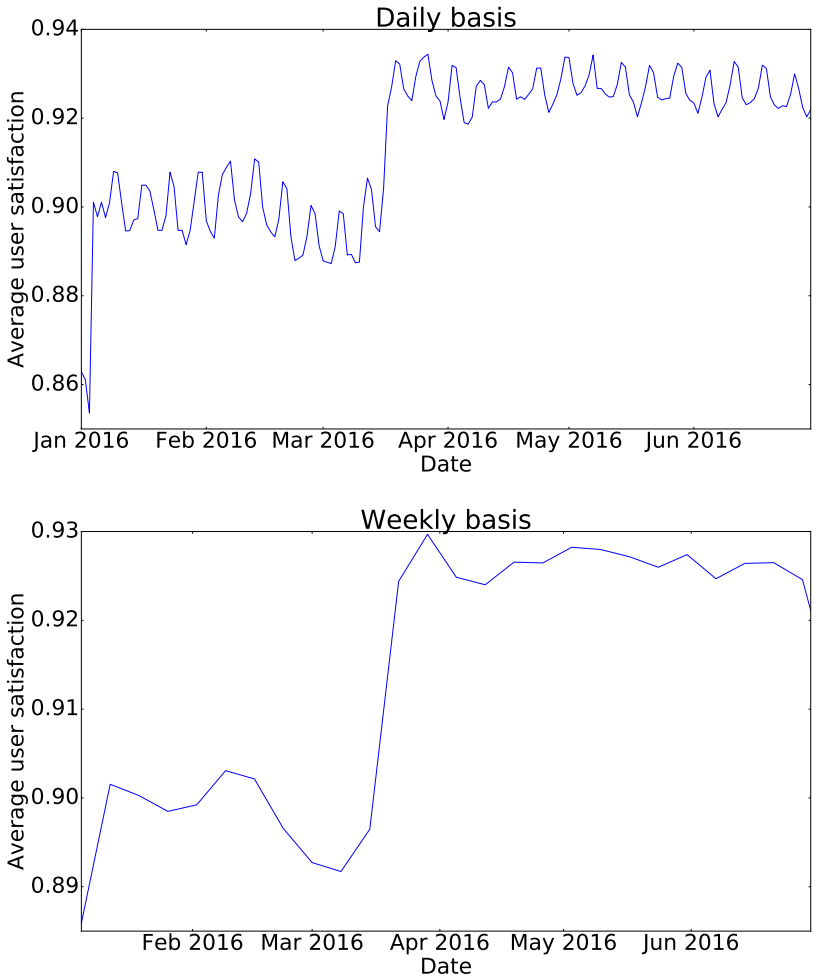

Figure 7.2: Average User Satisfaction: Daily and Monthly Basis

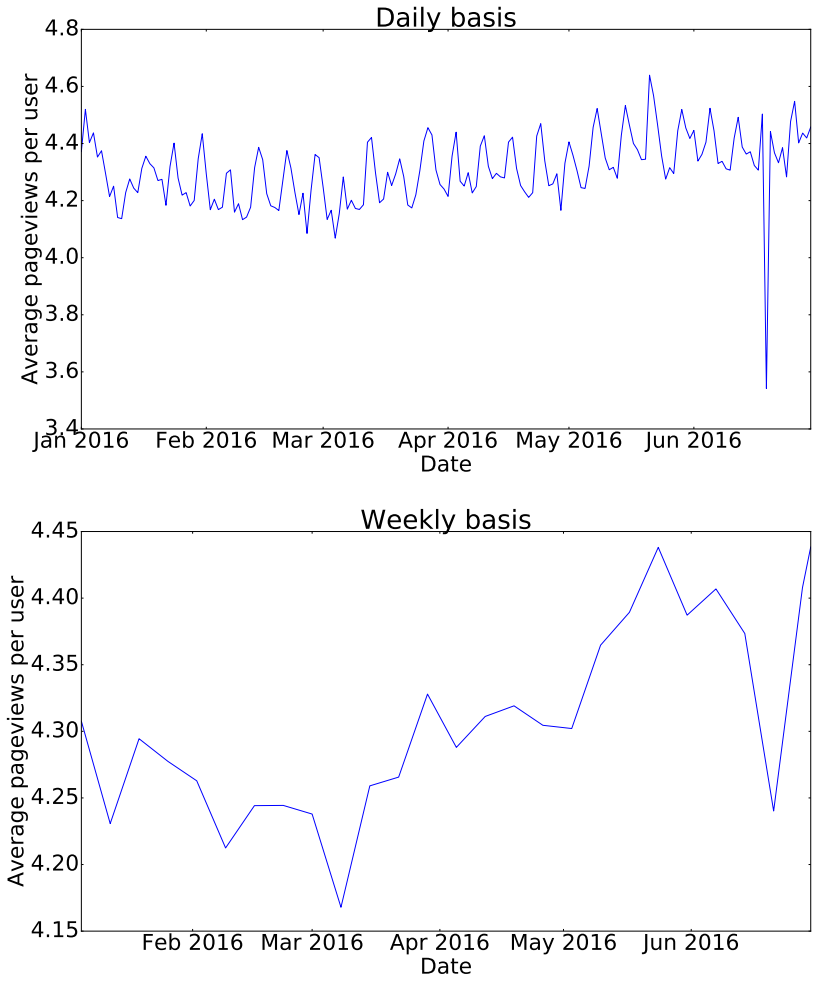

Figure 7.3: Average Transactions per User: Daily and Monthly Basis

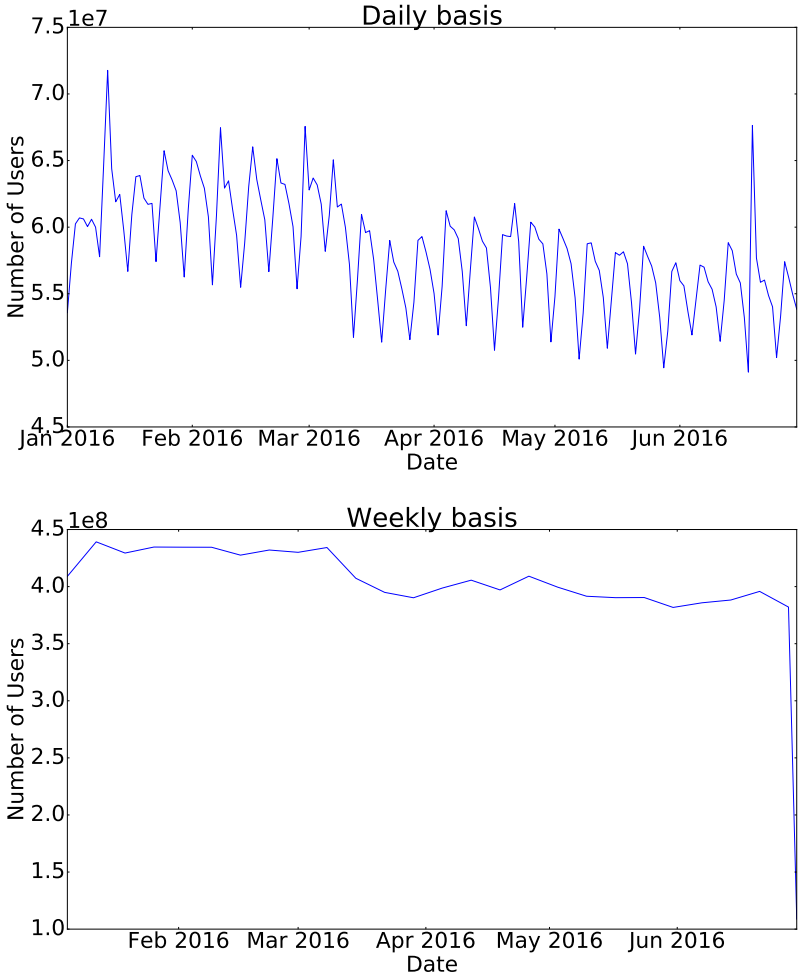

Figure 7.4: Average Users: Daily and Monthly Basis

By using the collected data, the obtained results were:

$$(rev_{exp} * satisf)_{day} = 8158348.87\$$$

$$(rev_{exp} * satisf)_{week} = 10527561.5\$$$

Depending on the chosen *basis*, the expected revenue varies due to the modelled rpu. In this case, the lowest value is taken, as it constraints the final result, by modelling the worst case situation. Therefore, henceforth $rev_{exp} * satisf = (rev_{exp} * satisf)_{day}$.

Then, *total_revenue* per topology $T^{\mu}_{(i)} \in \Lambda$ would be calculated by multiplying the previous result and the corresponding availability, calculated before, for our subset of viable topologies. Consequently, final results for *total_revenue* term are:

$$total\_revenue = \begin{cases} 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{1} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{2} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{3} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{4} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{5} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{6} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{7} \\ 8158348.87 * 99.9\% \approx 8150190.52\$ & for \quad T^{\mu}_{8} \\ 8158348.87 * 99.9\% \approx 8150190.52\$ & for \quad T^{\mu}_{9} \\ 8158348.87 * 99.9\% \approx 8150190.52\$ & for \quad T^{\mu}_{10} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{11} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{12} \\ 8158348.87 * 99.95\% \approx 8154269.70\$ & for \quad T^{\mu}_{13} \end{cases}$$

Complete donations at Wikimedia Foundation are considered in this evaluation as this organization does not provide its financial information divided by a concrete project and language. It is reasonable to assume a popularity division, per project and language, but it may not be the case always and some expenses may be related to many projects and could not be divided. However, if the popularity division was accepted, the revenue results for each topology would only be scaled by the same factor and the order would be preserved.

7.1.2 Cost

On the other hand, the second main term of the utility is the cost. Its concept is developed in section 6.3 as the aggregation of direct (instance pricing) and adaptation costs (e.g. requiring a database replica). In this case, one of the identified possible adaptation costs is the scaling

of a VM during November and December months, being able to cope with possible workload increments occurring when the English campaign is held. However, those months are not considered in the study.

Furthermore, supporting tools such as Nefolog[XA+13] help in the cost calculation process. However, all the considered providers offered its own cost calculator tools too[6]. Obtained *cost* for the considered topologies is:

$$cost = \begin{cases} 1759.32\$ & for \quad T_1^\mu \\ 2293.63\$ & for \quad T_2^\mu \\ 1377.05\$ & for \quad T_3^\mu \\ 6355.27\$ & for \quad T_4^\mu \\ 2662\$ & for \quad T_5^\mu \\ 3549.48\$ & for \quad T_6^\mu \\ 2070.78\$ & for \quad T_7^\mu \\ 3336.19\$ & for \quad T_8^\mu \\ 1893.98\$ & for \quad T_9^\mu \\ 5282.3\$ & for \quad T_{10}^\mu \\ 1016.5\$ & for \quad T_{11}^\mu \\ 1355.33\$ & for \quad T_{12}^\mu \\ 760.2\$ & for \quad T_{13}^\mu \end{cases}$$

7.1.3 Utility and Results

Previous sections 7.1.1 and 7.1.2 present the different evaluation terms *total_revenue* and *cost* for equation 6.1:

$$u(T^\mu, \Gamma, t_q, P) = total\_revenue(T^\mu, t_q, P) - cost(T^\mu, \Gamma, t_q, P)$$

considering a fixed subset of viable application topologies $T_{(i)}^\mu \in \Lambda$ introduced at table 7.1 during a time period Φ and some requirements.

By applying the previous equation, we can evaluate the different topologies and, aiming to a maximum utility, order the results which are comparable now because, as was commented in section 6.1, utility functions map objects (topologies in this case) to a number in $\mathbb{R}$ domain.

[6] http://calculator.s3.amazonaws.com/index.html, https://azure.microsoft.com/en-us/pricing/calculator/, https://www.rackspace.com/calculator and https://cloud.google.com/products/calculator/

Topology	Utility	Revenue	Cost
T^{μ}_{13}	8153509.50	8154269.70	760.2
T^{μ}_{11}	8153253.20	8154269.70	1016.5
T^{μ}_{12}	8152914.37	8154269.70	1355.33
T^{μ}_{3}	8152892.65	8154269.70	1377.05
T^{μ}_{1}	8152510.38	8154269.70	1759.32
T^{μ}_{7}	8152198.92	8154269.70	2070.78
T^{μ}_{2}	8151976.07	8154269.70	2293.63
T^{μ}_{5}	8151607.70	8154269.70	2662
T^{μ}_{6}	8150720.21	8154269.70	3549.48
T^{μ}_{9}	8148296.54	8150190.52	1893.98
T^{μ}_{4}	8147914.43	8154269.70	6355.27
T^{μ}_{8}	8146854.33	8150190.52	3336.19
T^{μ}_{10}	8144908.22	8150190.52	5282.3

Table 7.2: Utility results

Table 7.2 shows the numerical results of the utility evaluation:

$$u(T^{\mu}_{13}) \geq u(T^{\mu}_{11}) \geq u(T^{\mu}_{12}) \geq u(T^{\mu}_{3}) \geq u(T^{\mu}_{1}) \geq u(T^{\mu}_{7}) \geq u(T^{\mu}_{2}) \geq \\ \geq u(T^{\mu}_{5}) \geq u(T^{\mu}_{6}) \geq u(T^{\mu}_{9}) \geq u(T^{\mu}_{4}) \geq u(T^{\mu}_{8}) \geq u(T^{\mu}_{10})$$

Therefore, as explained in section 2.4.1, considering monotonic utility functions, this evaluation list maintains the preference order for the subset of viable topologies considered:

$$T^{\mu}_{13} \succ T^{\mu}_{11} \succ T^{\mu}_{12} \succ T^{\mu}_{3} \succ T^{\mu}_{1} \succ T^{\mu}_{7} \succ T^{\mu}_{2} \succ T^{\mu}_{5} \succ T^{\mu}_{6} \succ T^{\mu}_{9} \succ T^{\mu}_{4} \succ T^{\mu}_{8} \succ T^{\mu}_{10}$$

Results show that the preferred topology is one offered by Google Compute Engine, deploying MediaWiki front-end and back-end at two different off-premise instances. In fact, Google Compute Engine is the preferred provider. On the other hand, the last topology by preference is offered by Rackspace, deploying both components separately. However, it should be pointed out that, opposed to the general first thinking, the last topology at the ranking is not associated to the highest cost, which is derived from the DaaS solution. This supports the idea that, although it represents a big influence, basing the decision on how to deploy an application in the cloud only on expected monetary expenses may not be the best option. A balance between predicted revenue and costs through a utility-based analysis gives a more complex approach and may lead to an equilibrium among the interests of business and IT experts. Moreover, utility functions allow multiple definitions depending on the purpose and the web application under study. In this analysis only the number of user editions at English Wikipedia shows a possible relationship with the received donations and, therefore,

is included as an influence at users satisfaction. However, if for a particular case an specific metric is found to directly affect the revenue of the application, it can be included at the analysis by redefining the utility function.

8 Outcome and Future Work

The wide adoption of cloud computing has led to an increase on offerings available to the public. However, optimizing their use and configuration is essential both for providers (able to have more clients) and users (having less monetary expenses while managing their resources in an efficient way). For that reason, the deployment of web applications in the cloud should be carefully analysed. Multiple studies are focused on finding suitable evaluation methods. One of them considers that the balance between performance, revenue and costs is achieved through utility functions that take into account, among other metrics, the revenue model and some viable topologies for the application under study. They are derived through an individual component level characterization rather than considering it a whole. However, as it is domain and application specific, determining a revenue model is one of the challenges faced with this approach and real data should be collected and analysed to study its possible influences.

This document aims to evaluate a set of different cloud distributions for MediaWiki application by using utility-based analysis derived through real data of last years focusing on the economical trade-off between associated profits and costs and their influences. For that purpose, *Chapter 2* presents the fundamental concepts of cloud computing (e.g. deployment models) and gives an overview of some application topology languages such as Winery, easing the description of the components of a web application and its interrelations. This helps analysing the possible viable cloud distributions of an application. Moreover, the basic concepts of utility theory and python libraries used are also covered in this chapter, as well as [Fre16] thesis, which is the starting point for this document. *Chapter 3* summarizes some of already existing works that apply utility theory to cloud computing analysis. Each author defines an approach but they tend to find a maximization of a trade-off between a cost and a term which can be oriented from the economical or the performance point of view (e.g. CPU resources allocation). Furthermore, some existing works related to analysing wikis (concretely Wikipedia) and tools address for it (such as Wikibench) are mentioned.

Chapter 4 starts the examination with some of the revenue models applied nowadays at web applications, with examples, as the first step to give a general overview of state-of-the-art models. However, those are not unique but domain specific and a combination of them can be used depending on the case. Consequently, when considering a determined application such in this work, the application specific revenue model should be derived as well as studying its topology because, as it was mentioned before, it increases the amount of deployment possibilities. It is done in *Chapter 5*, which explains the derived revenue model applied obtained through real data metrics that were collected and analysed in case a direct relationship between them and received donations exist and needed to be included in the utility. The concept applied is concretely defined in *Chapter 6*, where each term is explained and the formulae are derived. They, as well as the derived monthly revenue model, are used at the evaluation performed at *Chapter 7* where a subset of possible viable topologies from

different cloud providers is analysed in order to choose the optimal solution for the migration. Results are also presented in the chapter and show that the balance between business and IT interests is better achieved by means of a more complex utility approach, which improves the one based only on predicted costs.

Future works in this area may follow the utility analysis with other web applications and derive new revenue and costs models for them. Considering not only the obtained monetary returns and derived operational costs, but also how the end-user satisfaction affects the utility of the application. Moreover, due to the limitations on available data, this analysis was performed in a fixed time interval, but future works can study more complex scenarios if past data records are higher. Cloud computing seems to have many applications and any *e*-technology or activity (e.g. IoT) may promote its use. Therefore, this area can have an impact on efficient future cloud uses both for cloud providers and clients, as well as in distributions or re-distribution of applications.

Bibliography

[ABLS13] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch. How to adapt applications for the Cloud environment. *Computing*, 95(6):493–535, 2013.

[ARSL14] V. Andrikopoulos, A. Reuter, S. G. Sáez, and F. Leymann. A GENTL approach for cloud application topologies. In *European Conference on Service-Oriented and Cloud Computing*, pages 148–159. Springer, 2014.

[ARXL14] V. Andrikopoulos, A. Reuter, M. Xiu, and F. Leymann. Design Support for Cost-efficient Application Distribution in the Cloud. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 697–704. IEEE, 2014.

[ASLW14] V. Andrikopoulos, S. G. Sáez, F. Leymann, and J. Wettinger. Optimal distribution of applications in the cloud. In *International Conference on Advanced Information Systems Engineering*, pages 75–90. Springer, 2014.

[Bar07] C. Barz. *Risk-Averse Capacity Control in Revenue Management (Ph. D. thesis)*. Springer Verlag, 2007.

[BBH+13] T. Binz, U. Breitenbücher, F. Haupt, O. Kopp, F. Leymann, A. Nowak, and S. Wagner. *OpenTOSCA - a runtime for TOSCA-based cloud applications*, pages 692–695. Springer Berlin Heidelberg, 2013.

[BBKL14] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann. TOSCA: portable automated deployment and management of cloud applications. In *Advanced Web Services*, pages 527–549. Springer, 2014.

[BBLS12] T. Binz, G. Breiter, F. Leymann, and T. Spatzier. Portable Cloud Services Using TOSCA. *IEEE Internet Computing*, 16(3):80–85, 2012.

[BGPCV12] L. Badger, T. Grance, R. Patt-Corner, and J. Voas. Cloud Computing Synopsis and Recommendations. *Recommendations of the National Institute of Standards and Technology, Tech. Rep*, 2012.

[BMT05] C. Bellettini, A. Marchetto, and A. Trentini. TestUml: user-metrics driven web applications testing. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1694–1698. ACM, 2005.

[BYV+09] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[DWC10] T. Dillon, C. Wu, and E. Chang. Cloud computing: issues and challenges. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 27–33. Ieee, 2010.

[Eva08] D. S. Evans. The economics of the online advertising industry. *Review of network economics*, 7(3), 2008.

[Fis88] P. C. Fishburn. *Utility theory*. Wiley Online Library, 1988.

[Foua] W. Foundation. Benefactors 2014-2015 - Wikimedia Foundation. `https://wikimediafoundation.org/wiki/Benefactors/2014-2015`.

[Foub] W. Foundation. Financial reports - Wikimedia Foundation. `https://wikimediafoundation.org/wiki/Financial_reports`.

[Fouc] W. Foundation. Wikimedia Foundation 2015-2016 Annual Plan.pdf. `https://upload.wikimedia.org/wikipedia/foundation/4/43/WMF2015-16AnnualPlan.pdf`.

[Fou16a] P. S. Foundation. The Python Standard Library — Python 3.5.2 documentation. `https://docs.python.org/3/library/`, Jul 2016.

[Fou16b] W. Foundation. Frequently asked questions of Wikimedia Foundation. `https://wikimediafoundation.org/wiki/FAQ/en`, March 2016.

[Fre16] F. H. Frech. Utility-based Analysis of Evolving Cloud Application Topologies. Master's thesis, University of Stuttgart, Jan 2016.

[Gra06] J. Gray. A conversation with Werner Vogels. *ACM Queue*, 4(4):14–22, 2006.

[H+07] J. D. Hunter et al. Matplotlib: A 2D graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[Hak70] N. H. Hakansson. Optimal investment and consumption strategies under risk for a class of utility functions. *Econometrica: Journal of the Econometric Society*, pages 587–607, 1970.

[IGC04] D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, pages 160–169. IEEE, 2004.

[Joh07] T. Johnson. Utility functions. *C2922 economics, Heriot Watt University, Edinburgh*, 2007.

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery: A modeling tool for TOSCA-based cloud applications. In *International Conference on Service-Oriented Computing*, pages 700–704. Springer, 2013.

[KPM] KPMG. Audit_Report_-_FY_14-15_-_Final.PDF. `https://upload.wikimedia.org/wikipedia/foundation/0/0b/Audit_Report_-_FY_14-15_-_Final.PDF`.

[LBRK10] S. Leimeister, M. Böhm, C. Riedl, and H. Krcmar. The Business Perspective of Cloud Computing: Actors, Roles and Value Networks. In *ECIS*, 2010.

[LL11] K.-L. Lin and C.-L. Lin. Applying utility theory to cost allocation of pavement maintenance and repair. *International Journal of Pavement Research and Technology*, 4(4):212–221, 2011.

[LS10] W. Lehner and K.-U. Sattler. Database as a service (DBaaS). In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 1216–1217. IEEE, 2010.

[MB12] C. Madhavaiah and I. Bashir. Defining cloud computing in business perspective: a review of research. *Metamorphosis: A Journal of Management Research*, 11(2):50–65, 2012.

[MF11] D. Minarolli and B. Freisleben. Utility-based resource allocation for virtual machines in cloud computing. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 410–417. IEEE, 2011.

[MJ98] D. Mosberger and T. Jin. Httperf—a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[Mor07] J. C. Morris. DistriWiki: a distributed peer-to-peer wiki network. In *Proceedings of the 2007 international symposium on Wikis*, pages 69–74. ACM, 2007.

[Nah04] F. F.-H. Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[NDDSD13] M. Z. Nkhoma, D. P. Dang, and A. De Souza-Daw. Contributing factors of cloud computing adoption: a technology-organisation-environment framework approach. In *Proceedings of the European Conference on Information Management & Evaluation*, pages 180–189, 2013.

[NST13] NSTA. Chapter 8: Correlation. *NSTA Reports*, 25(4):29, 11 2013.

[PDAL+09] N. Paton, M. A. De Aragão, K. Lee, A. A. Fernandes, and R. Sakellariou. Optimizing utility in cloud computing through autonomic workload execution. *Bulletin of the Technical Committee on Data Engineering*, 32(1):51–58, 2009.

[PS13] D. Palma and T. Spatzier. Topology and orchestration specification for cloud applications version 1.0. *OASIS Standard*, November 2013.

[PSW80] J. S. Pliskin, D. S. Shepard, and M. C. Weinstein. Utility functions for life years and health status. *Operations research*, 28(1):206–224, 1980.

[PWGB10] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 400–407. IEEE, 2010.

[Rap03] M. Rappa. Business models on the web. *Available at Managing the Digital Enterprise website: http://digitalenterprise. org*, 2003.

[RDG11] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 500–507. IEEE, 2011.

[RYF15a] X. Ruan, Z. Yin, and D. M. Frangopol. Risk matrix integrating risk attitudes based on utility theory. *Risk Analysis*, 35(8):1437–1447, 2015.

[RYF15b] X. Ruan, Z. Yin, and D. M. Frangopol. Risk Matrix Integrating Risk Attitudes Based on Utility Theory: Risk Matrix Integrating Risk Attitudes. *Risk Analysis*, 35(8):1437–1447, 2015.

[SAH+16] S. G. Sáez, V. Andrikopoulos, M. Hahn, D. Karastoyanova, F. Leymann, M. Skouradaki, and K. Vukojevic-Haupt. *Performance and Cost Trade-Off in IaaS Environments: A Scientific Workflow Simulation Environment Case Study*, volume 581 of *Cloud Computing and Services Science*, pages 153–170. Springer, February 2016.

[SAL16] S. G. Sáez, V. Andrikopoulos, and F. Leymann. Consolidation of Performance and Workload Models in Evolving Cloud Application Topologies. In *Proceedings of the 6th International Conference on Cloud Computing and Service Science (CLOSER 2016)*, pages 160–169, Rome, Italy, April 2016. SciTePress.

[SALS15] S. G. Sáez, V. Andrikopoulos, F. Leymann, and S. Strauch. Design support for performance aware dynamic application (Re-) distribution in the cloud. *IEEE Transactions on Services Computing*, 8(2):225–239, 2015.

[STFG08] J. D. Strunk, E. Thereska, C. Faloutsos, and G. R. Ganger. Using Utility to Provision Storage Systems. In *FAST*, volume 8, pages 1–16, 2008.

[Sti50] G. J. Stigler. The Development of Utility Theory. II. *Journal of Political Economy*, 58(5):373–396, 1950.

[Tho03] R. Thomson. The use of utility functions for investment channel choice in defined contribution retirement funds. II: A proposed system. *British Actuarial Journal*, 9(04):903–958, 2003.

[UPVS07] G. Urdaneta, G. Pierre, and M. Van Steen. A Decentralized Wiki Engine for Collaborative Wikipedia Hosting. In *WEBIST (1)*, pages 156–163, 2007.

[UPVS09] G. Urdaneta, G. Pierre, and M. Van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 53(11):1830–1845, 2009.

[vB09] E.-J. van Baaren. Wikibench: A distributed, wikipedia based web application benchmark. *Master's thesis, VU University Amsterdam*, 2009.

[VRMCL08] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[VT14] V. Varadharajan and U. Tupakula. Security as a service model for cloud environment. *IEEE Transactions on Network and Service Management*, 11(1):60–75, 2014.

[WTK+08] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer, and W. Karl. Scientific Cloud Computing: Early Definition and Experience. In *HPCC*, volume 8, pages 825–830, 2008.

[WTKD04] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 70–77. IEEE, 2004.

[XA+13] M. Xiu, V. Andrikopoulos, et al. The Nefolog & MiDSuS Systems for Cloud Migration Support. *Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Technical Report*, 8, 2013.

[XP09] K. Xiong and H. Perros. Service performance and analysis in cloud computing. In *2009 Congress on Services-I*, pages 693–700. IEEE, 2009.

[Yam13] S.-C. J. Yam. Decommercialization and anti-elitism: early years of Wikipedia 2001-2002. *International Journal of Arts & Sciences*, 6(1):533, 2013.

[ZL11] W. Zhuang and M. Z. Li. Revenue optimization of risk-averse managers with atemporal utility. *Journal of Revenue & Pricing Management*, 10(5):424–437, 2011.

[ZWS06] X. Zhu, Z. Wang, and S. Singhal. Utility-driven workload management using nested control design. In *2006 American Control Conference*, pages 6–pp. IEEE, 2006.

All links were last followed on September 27, 2016

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, September 27, 2016 _____

(María Elena Alonso Mencía)