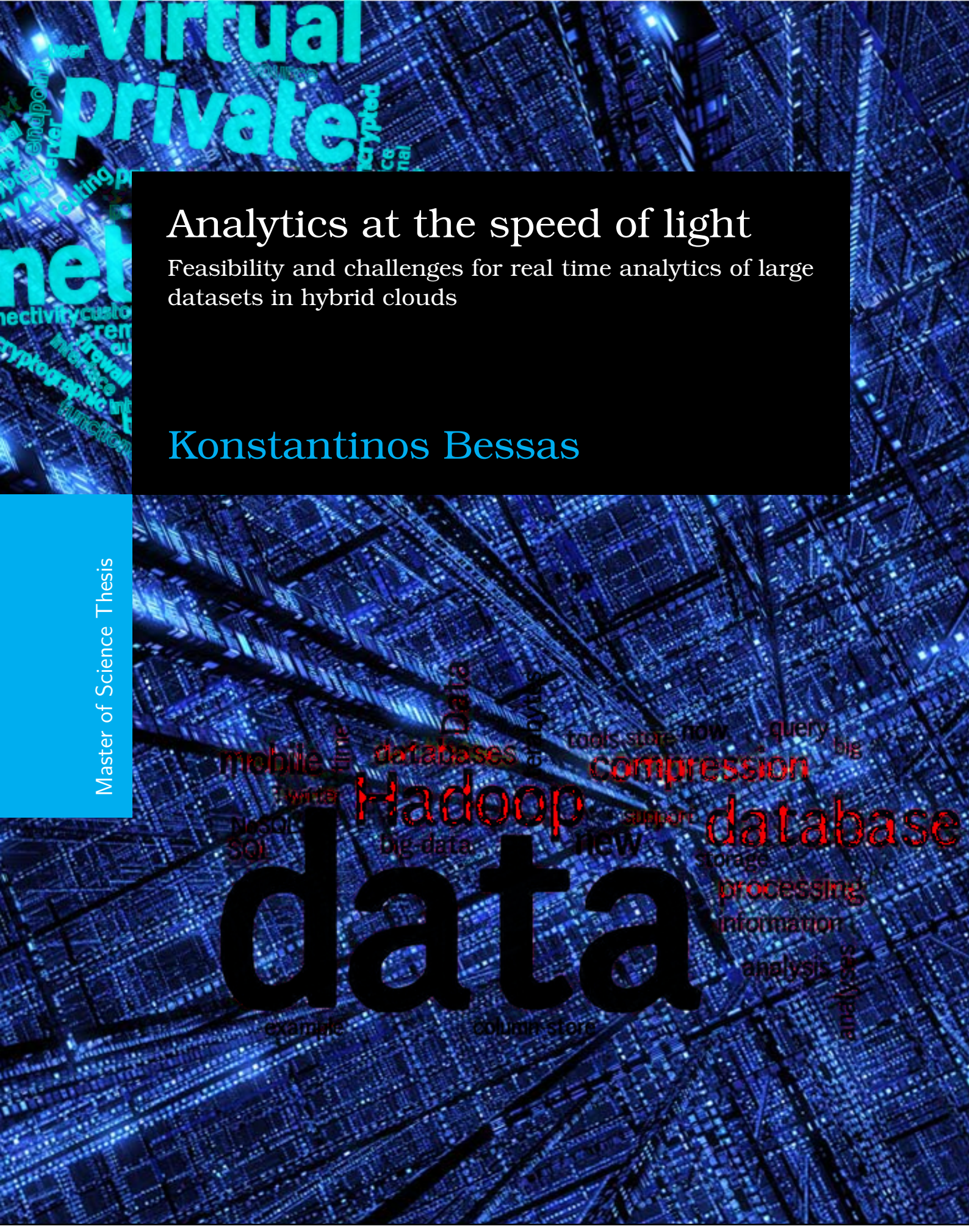


Analytics at the speed of light

Feasibility and challenges for real time analytics of large datasets in hybrid clouds

Konstantinos Bessas

Master of Science Thesis

TU Delft Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Analytics at the speed of light

Feasibility and challenges for real time analytics of large datasets in hybrid clouds

Master of Science Thesis

For the degree of Master of Science in Electrical Engineering (Track Telecommunications) at Delft University of Technology

Konstantinos Bessas

February 21, 2014

Faculty of Electrical Engineering, Mathematics and Computer Science · Delft University of Technology

This work was carried out on the Dutch national e-infrastructure with the support of SURF foundation.

Network Architectures and Services

Delft University of Technology
Department of Intelligent Systems
Network Architectures and Services group

Analytics at the speed of light

by

Konstantinos Bessas

Master of Science Electrical Engineering (Track Telecommunications)

Dated: February 21, 2014

Thesis Committee: Dr.ir. Fernando A. Kuipers

Dr.ir. Alexandru Iosup

Dr. Claudia Hauff

Abstract

"Real-time services" is a very challenging topic. Running analytics in real-time when there is an abstract network layer makes things even more complicated. The demand to analyze huge data-sets in real time or in the long term has been increasing over the past decade in many sectors including health-care, general science and various online services with a prime example being the trending Massively Multiplayer Online Games (MMOGs) community. Combining the network and computation infrastructure efficiently is a challenge that requires careful planning and deployment. This work extends the work that has been done in the field of cloud computing by incorporating the network infrastructure in the analytics procedure. We follow a threefold approach to the problem using mathematical analysis, simulations and real world experiments. The results have shown that real-time analytics over the network is feasible, despite the lack of QoS provisioning in many cases. The bottleneck of the total procedure is oscillating between the network and the computation part of the system, depending on the available computation and networking infrastructure as well as the time complexity of the algorithms used for the analytics.

Keywords: analytics, big data, cloud computing, network, real-time, data transfer, scheduling.

Table of Contents

Acknowledgments **xi**

1 Introduction **1**

2 Problem Description **3**

2-1 Real time analytics over a network 4

2-1-1 Parameters and notions 5

2-2 Scheduling of users' communication over the network 6

2-2-1 Parameters and notions 6

2-2-2 Objectives 8

3 Analytical Approach **11**

3-1 Introduction 11

3-1-1 Class of algorithm 11

3-1-2 Network link 12

3-1-3 Ratio of computation power 13

3-1-4 Datasets 14

3-2 One user, computations locally 15

3-2-1 Formulas 15

3-2-2 Computations 16

3-3 Multiple users, computations locally 16

3-4 One user, computations remotely using one cloud 18

3-4-1 Formulas 18

3-5 One User, computations remotely using multiple clouds 19

3-5-1 Formulas 20

3-6 Multiple users, computations remotely using one cloud 21

3-6-1 Formulas 21

3-7 Multiple users, computations remotely using multiple clouds . . . 22
3-7-1 Identical link rates . . . 23
3-7-2 Different link rates . . . 24
3-7-3 Formulas . . . 25
3-8 Comparison using realistic scenarios . . . 25
3-8-1 Single user . . . 25
3-8-2 Multiple users . . . 27

4 Simulation Approach **29**
4-1 Introduction . . . 29
4-2 Scheduling multiple users over a single link . . . 30
4-2-1 Simulation setup . . . 30
4-2-2 First in first out . . . 32
4-2-3 Equal shares . . . 33
4-2-4 Shortest job first - preemptive . . . 35
4-2-5 Shortest job first . . . 36
4-2-6 Other system metrics . . . 37
4-3 Scheduling multiple users over multiple links . . . 39
4-3-1 System setup . . . 39
4-3-2 Simulations with analytics complexity $O(n)$ . . . 43
4-3-3 Simulations with analytics complexity $O(n^2)$ . . . 43
4-3-4 Simulations with analytics complexity $O(n^3)$ . . . 45
4-3-5 Conclusion . . . 46

5 Real-world Experiment - Scheduler **47**
5-1 Introduction . . . 47
5-1-1 General information . . . 47
5-1-2 Technical information . . . 47
5-1-3 Benchmarking the private connection to the cloud . . . 48
5-2 Single link experiments . . . 49
5-2-1 System setup . . . 49
5-2-2 Deploying the simulation experiment on a real cloud . . . 50
5-2-3 Efficient dataset transfer using Secure Copy Protocol (SCP) . . . 52
5-2-4 Real time analytics using SCP . . . 55
5-2-5 Efficient dataset transfer over a private line using BBCP and UDT . . . 57
5-2-6 Real time analytics using BBCP and UDT . . . 60
5-3 Multiple links . . . 62
5-3-1 Real time analytics . . . 62

6 Conclusion and Future Work **67**
6-1 Conclusion . . . 68
6-2 Future work . . . 68

A Sara HPC Cloud **71**
A-1 General information 72
A-2 Technical information 72

B Simulation Models **73**
B-1 FIFO 74
B-2 ES 75
B-3 SJF-preemptive 76
B-4 SJF 77

Bibliography **79**

Glossary **83**
List of Acronyms 83

List of Figures

2-1 Abstract system setup. 4
2-2 Basic system setup showing the corresponing data-paths. 5
2-3 System setup with multiple clouds and connectivity options. 7
2-4 Function representation of the system. 8

3-1 One user, local computations. 15
3-2 Multiple users, local computations. 17
3-3 Markov Chain for shared pool of resources for multiple users locally. 17
3-4 Probability density of number of users with a dataset waiting to be sent for the "multiple users - local computations" case 18
3-5 One user, remote computations using one cloud. 19
3-6 One user, remote computations using multiple clouds. 20
3-7 Multiple users, remote computations using one cloud. 21
3-8 Markov Chain describing the network part of the multiple user scenario with one cloud. 21
3-9 Multiple users, remote computations using multiple clouds. 23
3-10 Markov Chain describing the network part of the multiple user scenario with multiple clouds and identical link rates. 23
3-11 System model of multiple user scenario with multiple clouds and different link rates. 24
3-12 Comparison of total response time for a variety of algorithmic complexities spots and one user. 26
3-13 Comparison of total response time for a variety of algorithmic complexities spots and multiple users. 28

4-1 Queue size throughout the FIFO queue simulation for various link capacities. . . 33
4-2 Active transfers throughout the ES queue simulation for various link capacities. . 34
4-3 Queue size throughout the SJFP queue simulation for various link capacities. . . 36
4-4 Queue size throughout the SJF queue simulation for various link capacities. . . . 37

4-5 Idle Time of the link for different allocation policies and link speed. 38
4-6 Total response time results. 39
4-7 Simulating multiple users over multiple links. 42
4-8 Queue size throughout the multiple clouds simulation for different link allocation policies ($O(n)$). 43
4-9 Cost for the multiple clouds simulation for different link allocation policies ($O(n)$). 44
4-10 Queue size throughout the multiple clouds simulation for different link allocation policies ($O(n^2)$). 44
4-11 Cost for the multiple clouds simulation for different link allocation policies ($O(n^2)$). 45
4-12 Queue size throughout the multiple clouds simulation for different link allocation policies ($O(n^3)$). 45
4-13 Cost for the multiple clouds simulation for different link allocation policies ($O(n^3)$). 46

5-1 Throughput benchmark results of our system using iperf. 49
5-2 System setup for real world experiment using one IaaS cloud. 50
5-3 Real world experiment using the same user setup as in the simulation approach and one Infrastructure as a Service (IaaS) cloud. 51
5-4 Efficient dataset transfer using one IaaS cloud over a private link (SCP). 53
5-5 Efficient dataset transfer using one IaaS cloud over the Internet (SCP). 55
5-6 Real time analytics experiment using one IaaS cloud over a private line (SCP). . 56
5-7 Real time analytics experiment using one IaaS cloud over the Internet (SCP). . . 57
5-8 Efficient dataset transfer using one IaaS cloud over a private line (BBCP). 59
5-9 Efficient dataset transfer using one IaaS cloud over a private line (UDT4). 60
5-10 Real time analytics experiment using one IaaS cloud over a private line (BaBar Copy (BBCP)). 61
5-11 Real time analytics experiment using one IaaS cloud over a private line (UDP-based Data Transfer (UDT)). 62
5-12 Various systems metrics over time for real time analytics experiment using multiple links (SCP). 63
5-13 Various systems metrics over time for real time analytics experiment using multiple links (BBCP). 64
5-14 Various systems metrics over time for real time analytics experiment using multiple links (UDT). 65

6-1 Total response time results for the setup of the simulation approach using all the different approaches studied throughput this thesis work. 67

B-1 FIFO queue simulation model. 74
B-2 ES queue simulation model. 75
B-3 SJFP queue simulation model. 76
B-4 SJF queue simulation model. 77

List of Tables

2-1 Objectives regarding the response time. 8
2-2 Objectives regarding cost. 8
2-3 Objectives regarding the system availability. 9

3-1 Settings used in linpack to carry out the benchmarks. 13
3-2 Performance of different computational spots. 14
3-3 Average computational time per dataset line in seconds. 14
3-4 Total computation time in "Small server" environment (seconds). 16
3-5 Test cases to compare the single user scenario. 26
3-6 Test cases to compare the multiple users scenario. 27

4-1 Hypothetical users of a datacenter and their correspondent dataset sizes. 31
4-2 Average dataset size in terms of bytes and number of lines. 32
4-3 dataset size distribution for the users considered for the multiple users over multiple links simulation. 42

5-1 Modified TCP settings used in both the server and the cloud to optimize throughput (all values are in bytes). 48
5-2 Parallel transfers per user for the experiment using the same user setup as in the simulation approach. 50
5-3 Deployed Virtual Machine(s) (VM(s))/Parallel Transfers - Optimal Dataset Transfer Experiment. 53

Acknowledgments

I would like to express the deepest appreciation to my supervisor and committee chair, Dr.ir. Fernando A. Kuipers for his help and for the trust he showed to me prior to completing this MSc thesis.

Additionally, I would like to extend my sincere gratitude to my supervisor and committee member Dr.ir. Alexandru Iosup, for his invaluable help and guidance.

I would also like to thank Dr. Claudia Hauff, for kindly honoring me with her presence in my graduation committee.

I am immensely grateful to my friends for the long conversations and comments on all the material concerning my thesis, but most importantly for their love and patience through difficult times.

Last but not least, I am especially grateful to my family and especially my parents, whose support over many years equipped me with the everlasting vision to move forward.

Delft, University of Technology — Konstantinos Bessas
February 21, 2014

Chapter 1

Introduction

Information technology (IT) has been increasing its share at the business over many years. IT has become pervasive in organizations and an inevitable key success factor in business. Companies, of all type and dimensions are growing the demand and invest with the objective of growing competitiveness in the market. Their goal is to reduce costs and simplify management of their technology while keep on providing more sophisticated services to their end users.

There have been network services since the invention of the internet in the 1970's. However in the last couple of years internet services offered online took on an even new dimension. One of the internet services that has flourished is the growing community of Massively Multiplayer Online Games (MMOGs) that has grown tremendously over the last few years. For the sake of showing this growth, let us take a look at some numbers behind *League of Legends*, a Multiplayer Online Battle Arena (MOBA) game with players from over 145 countries. With more than 70 million registered account names and more than 12 million daily active players[1], one can fully understand the importance of maintaining the network infrastructure and the computing power to host so many active players at once. Gigabytes of data are produced every second from such activity, some of which are very important for further analysis. Another example is the thousand web based game applications that exist on the internet, especially those within *Facebook*.

The requirement for real-time processing of high-volume data streams is pushing the limits of traditional data processing infrastructures. Cloud computing is a model for enabling ubiquitous, convenient and on-demand network access to a shared pool of configurable but virtual computing resources [1]. The definition of a hybrid cloud is also given there as "the cloud infrastructure composed of two or more distinct cloud infrastructures (private, community, or public) or that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability". Hybrid cloud can also mean the ability to connect collocation, managed and/or dedicated services with cloud resources. Many online services are part of a cloud computing system and are managed

[1]Riot Games, "League Of Legends: statistics", October 2012, http://majorleagueoflegends.s3.amazonaws.com/lol_infographic.png

with emphasis to the preferences, experience and behavior of the users. Over time, the accumulative increase in online services and their respective users has resulted in very big sets of data that, in many cases, need to be processed and analyzed quickly and efficiently. Only recently, Amazon has launched *Kinesis*[2], a service that promises to deal with real-time processing of streaming data at massive scale.

For what it is worth, MMOGs provide a fascinating new way of observing thousands of simultaneously socially interacting individuals engaged in all kind of activities associated with the gameplay environment [2]. This observation can actually be made by the data sets consisting of practically all actions of all players that can be stored - with literally minimal computational cost - in real time in the datacenters of the game server. The most efficient way to analyze all the data is by using the computational power of one or more cloud systems. The reason for this is the dynamic nature of computing resources that can be deployed. Analytics of large datasets require a diverse amount of time depending on the algorithms used and the information that needs to be retrieved. Doing this in the most efficient way is of utmost importance.

A common approach when analyzing a dataset is to model it as a graph. This is particularly important in complex networks as well as in online social network studies as it has been studied in [3]. There, the authors study the influence of of many different social link definitions with many different combinations. They use a threshold value to filter out links (relationships) of minor importance. Different values in this threshold may completely change the produced graphs which, for real time extraction and analysis of those graphs, could be computationally demanding.

In this thesis, we have focused on analysing potential systems that can deal with on-demand or streaming real-time analytics of big datasets. We have considered a centralized source for the data and we have studied different options for the computation and the network infrastructure that is required to process this data.

[2] http://aws.amazon.com/kinesis/

Chapter 2

Problem Description

There has been a great deal of research concerning the different provisioning and allocation policies available in cloud systems to find the best trade-off between cost and performance [4]. Performance evaluation of clouds and various ways and approaches to benchmark them have also been proposed and studied [5]. In order for this research to be complete, one should also include the overhead and operation cost that network links impose. There are various ways that the system could be built to work, depending on whether the data is critical and need to be processed on the fly, or if this can be done in a more cost efficient manner. There are trade-offs on the network part and various ways to achieve a working solution.

It is often the case that the data produced is not located anywhere near the computing cloud that will be used to process and analyze the data. Keeping data on the cloud itself actually costs a lot. This means that the data needs to traverse over the network to reach the destination (Amazon EC2 service, GoGrid, Google Cloud platform, etc.) whenever there is information that needs to be processed. In [6], the authors have addressed the problem of scheduling job executions and data movement operations in a distributed "Data Grid" environment. Their system set up includes the case that the data is already located remotely and the system replicates data in various remote spots at random intervals, which is in general of no interest to us.

In [7], an attempt to decide which workloads to outsource when there are multiple cloud providers has been made. They focus on minimizing the cost of the operations and do not study more objectives for their scheduler. Their approach is purely analytical and any experiments focus purely on the cloud in terms of cloud benchmarking. Although they have included the network part, their work does not include big sets of data, which will have a great impact on the response time[1] and will also affect the scheduling decisions greatly. An in-depth analysis with scheduling heuristics was attempted in [8]. It can be considered a more sophisticated approach of [7] with a deeper analysis of the network still absent. In [9], the authors have proposed an optimal Virtual Machine(s) (VM(s)) placement (OVMP)

[1] We will be using the term response time throughout this thesis as a metric for the total amount of time that is required to get the results back.

algorithm to provision the resources offered by multiple cloud providers. What is missing is how to schedule the communication of multiple users, and then split their workload in multiple cloud systems according to their best interest and needs.

Based on the previous, the following research questions apply:

1. What is the fraction of data that should be transferred to maximize system efficiency?
2. How to schedule the communication between the different interfering users of a datacenter and one or multiple Infrastructure as a Service (IaaS) cloud services?

Finally, it is also important to show evidence that the proposed approaches work in practice.

In the following subsections a more in-depth view of the above questions is given, along with various parameters and notions that influence the system. Many of the following need to be reviewed as the system is researched in the sections to follow.

2-1 Real time analytics over a network

Let us consider a datacenter where data is produced and stored by the activity of one or by multiple activities from various users. The question is: "Is it worth transferring the data over the network to a cloud and run the analytics there or is it more efficient to run them locally?"

In Figure 2-1, one can see a simple and abstract system setup that contains a datacenter, a cloud and the network connectivity between the two.

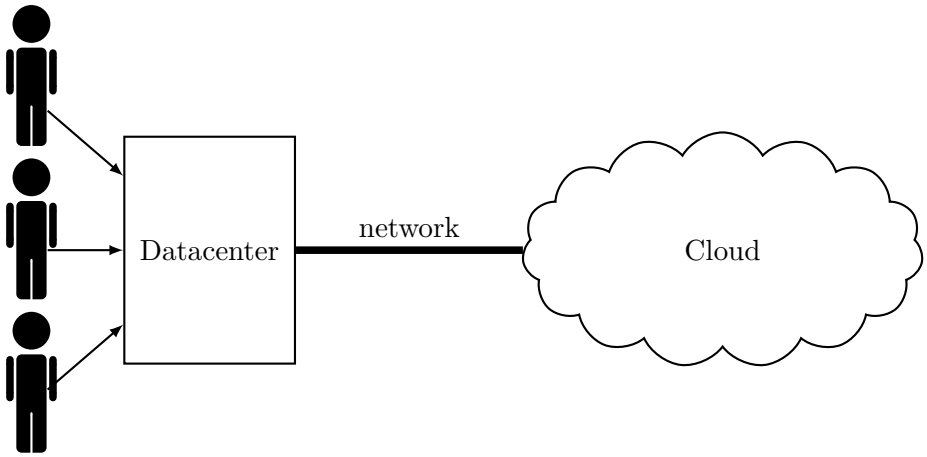


Figure 2-1: Abstract system setup.

There are fundamental differences in the resources available locally (datacenter) and those available remotely (cloud). From now on the terms local and datacenter, as well as the terms remote and cloud will be used interchangeably. The datacenter is considered to have some, but very limited computing power compared to the cloud. The resources that are available locally correspond to the hardware capabilities of the datacenter, while, on the cloud, the resources are virtual and can be replicated by means of running many instances.

If we consider that the data can be processed directly, without any intermediate steps of manipulation, then the time to run any analytics locally is the time to analyze the data on the local resources. On the other hand, when running the computations on the cloud, we need to sum the time that the data needs to traverse the network, the time needed to run the computations remotely and finally the time to get the results back. Many factors may affect the time of the computations, when running the computations on the cloud, which include time to set up a light-path or time to start and initialize instances. The response time for either local and remote computations is shown schematically in Figure 2-2 as the sum of the individual operation times t. The small r and the capital R correspond to the differences in computational resources between the two spots.

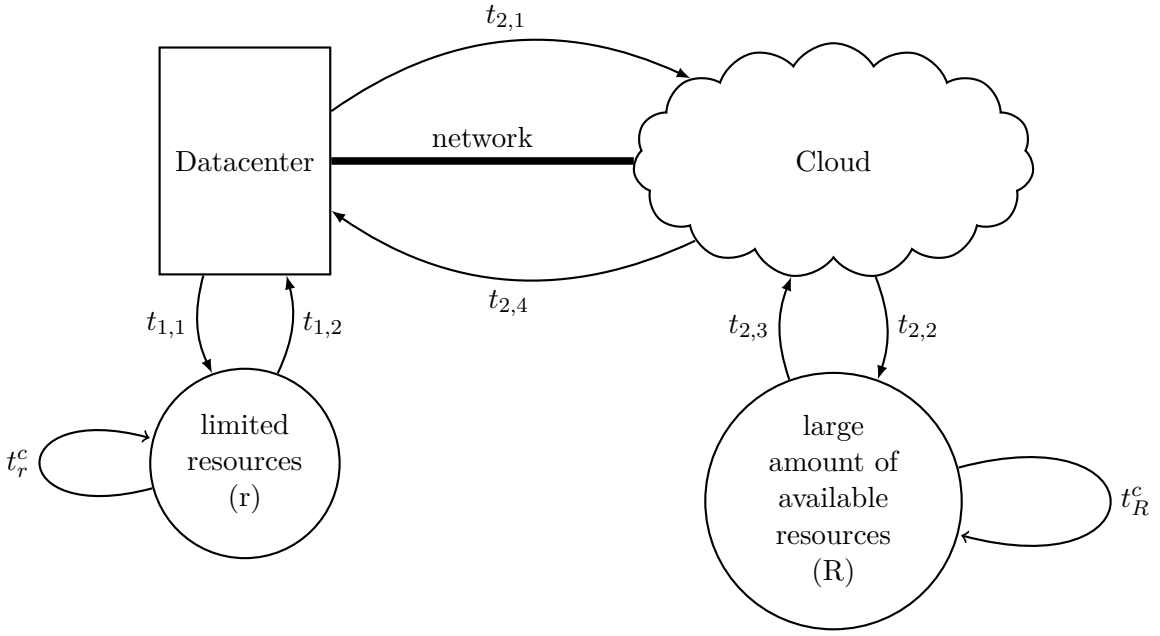


Figure 2-2: Basic system setup showing the corresponing data-paths.

2-1-1 Parameters and notions

As explained above, the system we are looking into can be seen as a three part system consisting of the following:

- Datacenter(s) (local computation spot and data origin)
- Network
- Cloud(s) (remote computation spot)

A set of characteristics is required to describe the system formally. By looking at each of the parts separately, there are many aspects to take into account.

The datacenter is characterized by the available computation power, as well as the amount of data waiting to be processed. The computation power is something abstract, which means that a notion to describe the amount of computation needed per input is required. The amount of data waiting to be processed is considered to be known at any point.

On the network part, there are many metrics available to characterize the overhead that is imposed. Some rather important for this particular case are latency, throughput and jitter. Other important factors could be possible losses or variations on the total available throughput, when the network is not a dedicated line but rather the Internet itself, or technical information on the ability to schedule multiple streams of data. Another important matter is the parallel utilization of the network (or part of it) by many users belonging to the same or multiple datacenters.

The cloud itself is characterized by its computation power (per instance), the amount of instances that a user can utilize and the cost of running all the different instances.

All the above information, combined with the knowledge of the complexity[2] of the algorithm that will be used to analyze the data, can be used as an input to a function that will present as an outcome whether it is worth sending the data to the cloud for remote computation, or if it is better to process the data locally. Thus, the previous problem can be considered as solvable.

2-2 Scheduling of users' communication over the network

Let us consider a datacenter that is utilized by multiple users. These are not end-users, but rather companies themselves, for instance Massively Multiplayer Online Games (MMOGs) hosters. These users want to send their data over a network to the cloud (could be a selection from multiple clouds) for analysis. The question is: "Which is the optimal way to schedule the transfer among the different users and where to send the data?"

A simple schematic of a possible system setup can be seen in Figure 2-3. More available options could be present (greater number of clouds, multiple dedicated connections among the computation spots, etc.).

This system is more complicated than the previous one in many aspects. As stated above, many users are hosted by the same datacenter. A lot of questions arise when all these users have to be scheduled in terms of when and how to transfer their data remotely. Which one of them to send at a particular time? In a dynamic setup, which users need to be stopped and for how long to serve users that may have higher priority? How to split the workload of the different users' data in multiple clouds depending on possible Quality of Service (QoS) characteristics [10] that every one of them has chosen?

2-2-1 Parameters and notions

Let us start from scratch. The system can be fully described by all the different parts. There is one single datacenter that is considered to host multiple users. All of these users have data

[2]Throughout this thesis, we will be referring using complexity to the time complexity of the algorithms used for analytics.

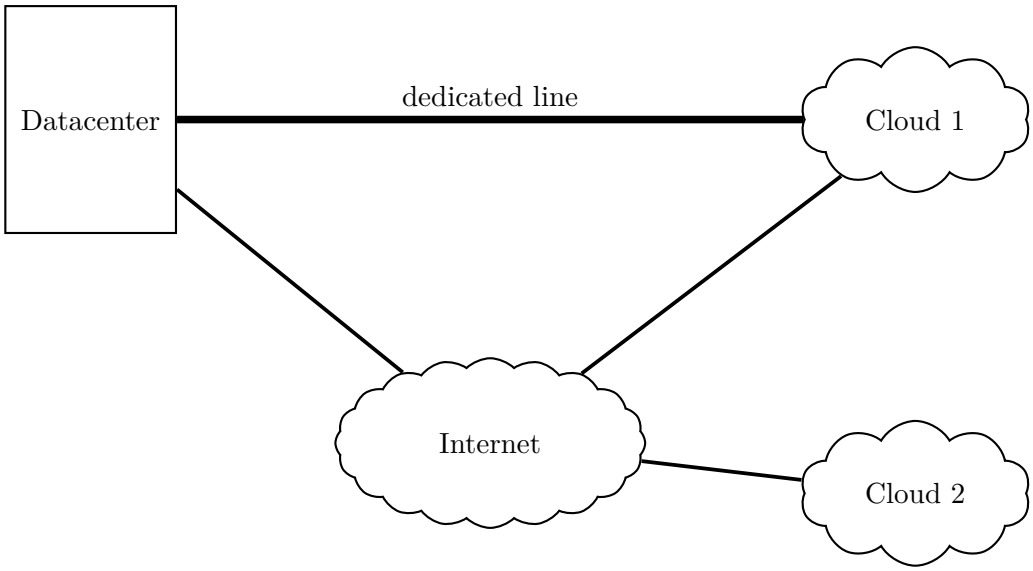


Figure 2-3: System setup with multiple clouds and connectivity options.

that they need to process. If there are free resources locally, the results from the previous section could apply. In any other case, the data needs to be transferred remotely. It is evident that the amount of users in the datacenter in combination with their corresponding job size is of great importance. Whenever there is data to be sent, the amount of data each user has to send, measured in any kind of data units (i.e. bytes, records, etc.), is very important. Finally, when the data reaches the cloud, the cpu-hours (cpuh) per data unit is needed.

For the cpuh, it is important to know what kind of analytics is going to take place, meaning the class of the algorithms in terms of time complexity that will be used, for instance $O(n)$ or $O(n^2)$ where n is the data input size (see Eq. 2-1). Different clouds may also have different computation power per instance, but in order to simplify the scheduler this may be omitted. In many cases, clouds offer a reduced price for pre-ordered resource and time usage up to a certain amount, which is of great importance to keep costs to a minimum, and is a key aspect when selecting the cloud to send the data when multiple options exist.

Depending on the load of the cloud at a given time, users do not always get the performance per VM(s) that is advertised. Cloud measurements have shown that the computation time differs greatly from time to time under the same experimental setup and can be up to ten times as much as the theoretical maximum [11]. Moreover, there is a setup time to take into account regarding the boot up of the VM(s) that will be used to carry out the computations. All the above can be summed up in Eq. 2-1.

$$T_C = \frac{C_A}{C_P} \times c_i + s(k) \tag{2-1}$$

where T_C corresponds to the computations time, C_A to the algorithmic complexity and C_P to the available computation power. c_i is an interference parameter from the total activity in the cloud and scales from 1 to 10. This can only be selected randomly as the cloud load cannot

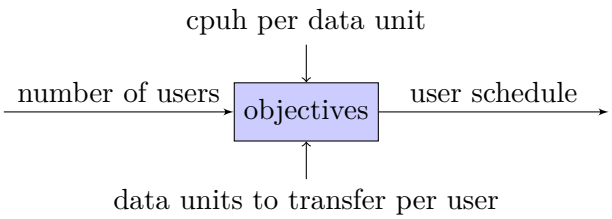


Figure 2-4: Function representation of the system.

Scheduler objective: Response time	
	absolute value
	a value below a certain threshold
	most values below a certain threshold

Table 2-1: Objectives regarding the response time.

Scheduler objective: Cost (endowed datacenter)	
	total cost below a certain value
	users with different budget

Table 2-2: Objectives regarding cost.

be known or predicted a priori. One of the reasons is that public IaaS service providers do not inform their customers on system availability and real time performance.

The VM(s) setup time parameter is a function of the number of VM(s) that the customer needs to boot up in parallel. Throughout the analytical part, the function that will be used for the setup time is $s(k) = \log_2 k \times c$ where k is the number of VM(s) that is going to be launched and c is a constant.

If we try to represent the above problem as a function, then it would look like Figure 2-4.

2-2-2 Objectives

There are multiple objectives that one can take into account. Minimizing the response time could be one of them. Response time is the total time needed to get the results back from the moment that the whole process is initiated and it is a key aspect in real time analytics. It can be defined in many alternative ways as shown in the examples in Table 2-1.

Another objective could be to reduce the cost. This objective can also be defined in multiple ways regarding whether we are interested in the total cost, or the cost of a particular user of the datacenter (see Table 2-2).

The availability is an important objective as well. A definition of various factors that influence the network availability as well as algorithms to measure it have been proposed in [12]. Although cloud services are generally offered with 99.99% uptime guarantee, the scheduler should take into account the system's availability and handle the requests of the users respectively. The availability also includes the various network links that connect the datacenter with the different cloud systems (see Table 2-3).

Scheduler objective: Availability	
	split the workload to all cloud systems to increase response rate
	send the same datasets to multiple clouds to decrease the network failure rate and opportunistically find the best execution time

Table 2-3: Objectives regarding the system availability.

A combination of the above objectives is also feasible but rather complicated. In case of single objectives, the scheduler can provide a working solution thus making the problem solvable. How close every solution will be to the optimal one and whether it is feasible to have an optimal solution are questions that actually need to be answered. The following sections will shine some light on everything discussed above.

Chapter 3

Analytical Approach

3-1 Introduction

From the previous section, it is straightforward that the system can be set up in various ways. It is not only the number of datacenters or clouds that affect the system, but also the different connectivity options among the various computation spots. And then, there are the different users that feed the system in different ways.

The following elements are of great importance to approach this system, as they greatly affect the performance of the whole process of analysis, meaning the transferring of the data and the computation time for the analytics to take place:

- Class of algorithm to be used for the analytics
- Network link between the datacenter and the cloud
- Ratio of computation power between the datacenter and the cloud

Let us take every single of the previous characteristics into consideration.

3-1-1 Class of algorithm

We will use asymptotic notation primarily to describe the running times of algorithms. The complexity classes of algorithms[1] that are of interest to us are those that usually describe the algorithms used to run the analytics on gaming data. These classes are $O(n)$, $O(n^2)$ and $O(n^3)$, which means that they have an asymptotic upper bound. The definition of the big-O notation is given below [13].

[1] Complexity is the only algorithm classification of interest to us and we will be referring to that through this thesis.

Definition 1. $O(g(n)) = \{f(n) : \textit{there exist positive constants } c \textit{ and } n_o \textit{ such that } 0 \leq f(n) \leq cg(n) \textit{ for all } n \geq n_o\}$

Mathematically, those classes can be depicted as:

$$T(n) = a \times n + C_c + c = O(n) \tag{3-1}$$

$$T(n) = a \times n^2 + C_c + c = O(n^2) \tag{3-2}$$

$$T(n) = a \times n^3 + C_c + c = O(n^3) \tag{3-3}$$

where n corresponds to the input size of the datasets and a to the computation time per iteration in the algorithm of interest. C_c are the complexity classes with order smaller than the one we are taking into account in every case. In the sections to follow, we have considered the best worst-case scenario and thus, those C_c classes have been excluded from the mathematical formulas that will be presented.

Using the input size in terms of file size (bytes) is not corresponding to the algorithmic complexity. For this reason, we had to come up with another notation that would illustrate the time complexity coherently. The datasets that we have been considering are usually big matrices with various columns and a very big amount of lines that correspond to the data for the different end-users. Finally, the algorithms used for the analytics usually iterate over the lines.

All the above gave birth to the computation time per element that will be used throughout this thesis. Since this time varies from machine to machine, we have to compute it each time. For this reason, a predefined set of datasets will be used from which an average computation time per element will be computed for the several (virtual or native) machines that we used for experimenting purposes. The previous can be classified and generalized in order to be used for any type of datasets regardless of its size in terms of columns and rows. With this information, the computation time of any dataset can be predicted very accurately leading to better scheduling decisions.

3-1-2 Network link

For the network connectivity, we have considered widely used practices present in similar systems that are already operational. We have considered 10 Mbit and 100 Mbit links when the connections are through the internet, with multiple connections being also an option [14]. Similarly, we have considered the usage of 1 Gbps and 10 Gbit dedicated links between the datacenter and the cloud.

For input of size n the time for the data to go across the link is:

$$t = \frac{n}{T} \tag{3-4}$$

where T corresponds to the nominal throughput of the network link and n to the input size of the data in terms of bytes.

3-1-3 Ratio of computation power

The difference of computation power between the computation spot that the data is generated and possibly analyzed and another more powerful one, can be defined as a ratio of the respective computation powers as defined in Equation 3-5.

$$\frac{CP_{DC}}{CP_C} = \frac{1}{N} \tag{3-5}$$

where CP_{DC} corresponds to the computational power available in the datacenter and CP_C to the sum of the computational power from all the different resources allocated in the cloud (i.e. the number of instances that one wants to deploy on the cloud). For this primitive approac,h the cost of the various elements of the system, like the network link and the cost of using the available computational power locally or in the cloud, will not be considered. N will be a number in a certain range.

To be precise, the actual ratio needs to be calculated for each datacenter-cloud pair, by running certain experiments using the exact analysis tools that are going to be used. The reason for this is that the various CPUs have differences in their architecture with the most important being their instructions set, parallelism, usage of memory caches, pipe-lining techniques and more. Not to mention that in this case, we are dealing with the actual hardware infrastructure on the datacenter spot (virtualization options may be available as well), while on the cloud the resources are virtual. For this reason the actual estimate is very difficult to be generated. This means that for accuracy in calculations of the ratio, one needs to measure the computation time per iteration instead of trying to deal with the complexity.

In order to get benchmarking results, we used *linpack*[2] to measure the performance of a typical datacenter rack. For the cloud part, we have utilized resources that have been available to us at an HPC Cloud facility located in Amsterdam which is available to the dutch academic community. More information can be found in Appendix A. A free tier from Amazon EC2 service has also been used for comparison purposes. Using the same setup as in the datacenter rack, we ran *linpack* on other native machines as well as on virtual machines on the two Infrastructure as a Service (IaaS) cloud environments mentioned in Table 3-1. The results are summarized in Table 3-2.

Number of equations to solve (problem size)	20000
Leading dimension of array	20000
Number of trials to run	8
Data alignment value (in Kbytes)	4

Table 3-1: Settings used in linpack to carry out the benchmarks.

The fragmented mathematical models that have been proposed throughout the literature cannot perfectly describe the complicated version of this system, which means that certain approximations need to be made. Every step to be taken will be explained and the reasons that certain approaches have been used will be described as we progress through this section.

[2] Intel®Math Kernel Library - http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download

Computational spot	CPUs	Frequency (GHz)	Cores	Threads	Performance (GFLOPS)
Datacenter	2	2.499	12	24	182
Small Server	2	2.997	2	4	9.8
Workstation	1	2.593	4	8	58.5
Sara cloud (“small”)	1	2.127	1	1	6.2
Amazon cloud (“t1.micro”)	1	1.799	1	1	1.9

Table 3-2: Performance of different computational spots.

3-1-4 Datasets

Datasets, as the word explains itself, are sets of data usually generated from computations of intense nature or user activity. In the context that is of interest to us throughout this thesis, datasets are files of different size, most often containing plain text, that include all the information that is gathered from a particular activity. This activity could be, for instance, the logging files of all the actions and decisions of users playing a particular game online. We have considered this definition of a dataset for all the sections to follow.

The usual layout of a dataset is a text file that contains rows and columns. The various lines correspond to the different users for whom the logging procedure is taking place, and the different columns to the different information gathered for every user.

Using the above definition of a dataset, the algorithms used for the analytics that read this dataset and extract all the necessary information can be considered that have a time complexity directly relevant to the number of lines that each dataset consists of. We have used a program that calculates the time of the calculations for different datasets for various algorithmic complexities (i.e. $O(n)$, $O(n^2)$ and $O(n^3)$). The results are shown in Table 3-3. A variety of datasets of different size were used for the calculations to verify the results.

Computational Spot	$O(n)$	$O(n^2)$	$O(n^3)$	Average comp. time per core per GFLOP
Datacenter	0.00000022	0.00042	0.802	0.000000007
Small Server	0.00000051	0.00103	2.133	0.000000043
Workstation	0.00000022	0.00042	0.8101	0.000000020
Sara cloud (“small”)	0.00000023	0.00067	0.982	0.000000027
Amazon cloud (“t1.micro”)	0.00000141	0.0029	6.003	0.000000196

Table 3-3: Average computational time per dataset line in seconds.

To calculate the average computational time per core per GFLOP, the theoretical maximum GFLOPS were used. We have considered that the analytics algorithm is not optimized for multi-processor usage and thus the theoretical maximum GFLOPS per CPU core was used [15]. Still there is a great difference among the different computational spots. The reason for this is the fact that the analytic algorithms may not utilize the processor to the maximum of its potential, as well as cases of different CPU scaling usage. Thus the computation time per core per GFLOP is not a reliable mechanism to use.

In the following subsections, different system setups will be approached from an analytical

point of view. For the reasons stated above, we have introduced a variable t_C (computational time per dataset line) which is different for every computational spot.

3-2 One user, computations locally

It was explained in the previous chapter that users utilize the datacenter to run the services they want to provide to their end-users. From these operations, a big amount of data is produced on an hourly basis or even more frequently. As it was stated, a very good example of this operation is Massively Multiplayer Online Games (MMOGs) companies that utilize the datacenter to run their game servers.

Apart from the services the users of the datacenter provide to their customers, they are also able to utilize the datacenter for the purpose of analyzing the data that are produced from the main service. This could be a very long and demanding operation that may even lead to resource exhaustion in regard to the main service. On the other hand, it is a very cheap way to analyze the data, especially if their size is insignificant, which means that the computation time will be short.

To explain this particular system formally, one can start from the different components shown in Figure 3-1. There is one user in the datacenter and the analytics for the data produced are ran locally. t_L^c corresponds to the time that is needed for the computations to finish.

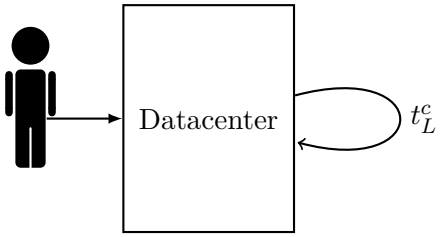


Figure 3-1: One user, local computations.

3-2-1 Formulas

The following formulas apply for the total response time for the system described above, depending on the algorithm used for the computations.

$$t_L = t_L^c \times k \tag{3-6}$$

$$t_L = t_L^c \times k^2 \tag{3-7}$$

$$t_L = t_L^c \times k^3 \tag{3-8}$$

where t_L^c corresponds to the computation per element in the datacenter and k corresponds to the dataset size in terms of elements, i.e. the iterations the algorithm runs on the datasets which could be the different lines or columns on the dataset itself. Throughout this thesis we have considered the usage of a dataset line as the element base for the calculations.

The previous equations refer respectively to $O(k)$, $O(k^2)$ and $O(k^3)$ classes of algorithms for the given input of data n which corresponds to a given amount of k elements.

3-2-2 Computations

A few computations based on the equations above can be seen in Table 3-4. Since it is quite straightforward to compute the computation time t_L, in Table 3-4 there is a comparison between the results from the equations above and actual analytics for the given datasets. For the experiments we have considered the "Small server" environment for which we have considered that we can utilize all CPU cores simultaneously.

Complexity / Size - #Lines	$O(n)$ (equation)	$O(n)$ (experiment)
1 GB - 20,029,940	5.10	5.29
1 GB - 27,843,152	7.10	7.44
10 GB - 200,324,210	51.08	51.91
100 GB - 2,954,854,506	753.48	766.20
1 TB - 7,348,595,000	3747.78	3771.89

Table 3-4: Total computation time in "Small server" environment (seconds).

The experimental results are extremely close to the results produced from the equations. For the computation time per input line we have used the results from Table 3-3. It is also straightforward here why the usage of computational power per byte is not recommended. As we can see from the first two lines of Table 3-4, the computational power per byte method would have produced the same execution time for the analytics procedure which would be far from true.

3-3 Multiple users, computations locally

In the common case, a single datacenter hosts multiple clients (Figure 3-2). Every client is assigned different racks and, in general, separate equipment. This means that the computation power is not affected by the presence of other users in the datacenter. Thus the system of the previous subsection applies here, for each user separately.

If we consider that a common pool of hardware resources is offered by the datacenter and is dedicated for the whole purpose of dataset analytics, then the computation time increases depending on the amount of time that the resources are shared among multiple users.

Users arrive in the system with a certain arrival rate and leave with a certain service rate. Thus, this system can be modeled using a finite Markov chain. Every state of the Markov chain corresponds to the number of users that are utilizing shared system resources for dataset analytics.

Here, we consider λ to be the arrival rate of a single user. For every state, the arrival rate for the next state to the right is the arrival rate of the single user multiplied by the remaining number of potential users that are available to appear. For the service rate, we consider

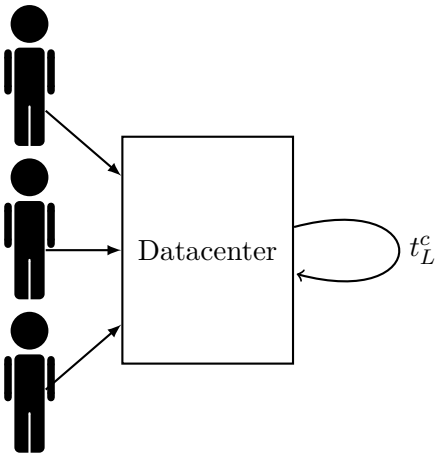


Figure 3-2: Multiple users, local computations.

it to be the rate of the user with the smallest dataset, in terms of a factor affecting the computation complexity, that is served at that point. The resulting Markov Chain can be seen in Figure 3-3.

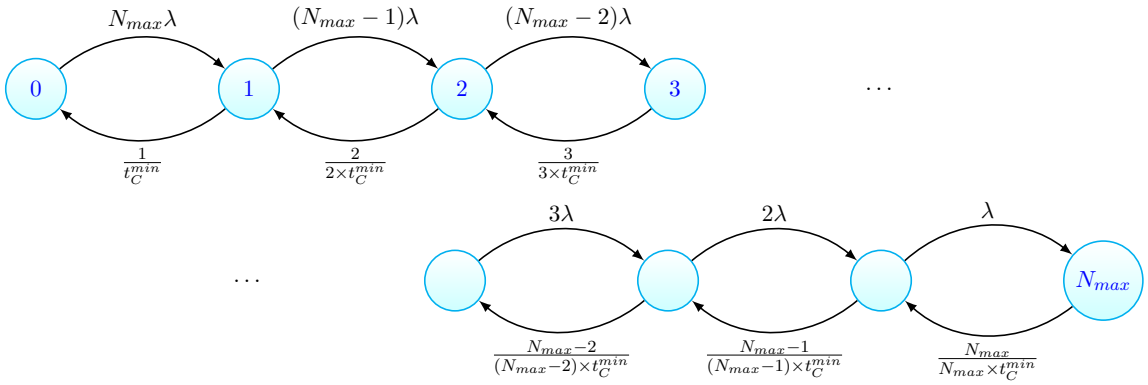


Figure 3-3: Markov Chain for shared pool of resources for multiple users locally.

We have considered for the *service* transitions in the Markov Chain the usage of the service time of the user at the system with the minimum remaining computation time at any taken system snapshot.

In [16] the general form of the steady state probabilities for the birth-death process was calculated. Using Equation 3-9 one can compute the steady state probabilities and thus the average number of users that occupy the system.

$$\pi_j = \frac{\prod_{i=0}^{j} \frac{\lambda_i}{\mu_{i+1}}}{1 + \sum_{k=1}^{\infty} \prod_{i=1}^{k} \frac{\lambda_i}{\mu_{i+1}}} \tag{3-9}$$

After approximating the minimum service time in every state with an average service time $E[t_c]$, we come up wit the plot shown in Figure 3-4.

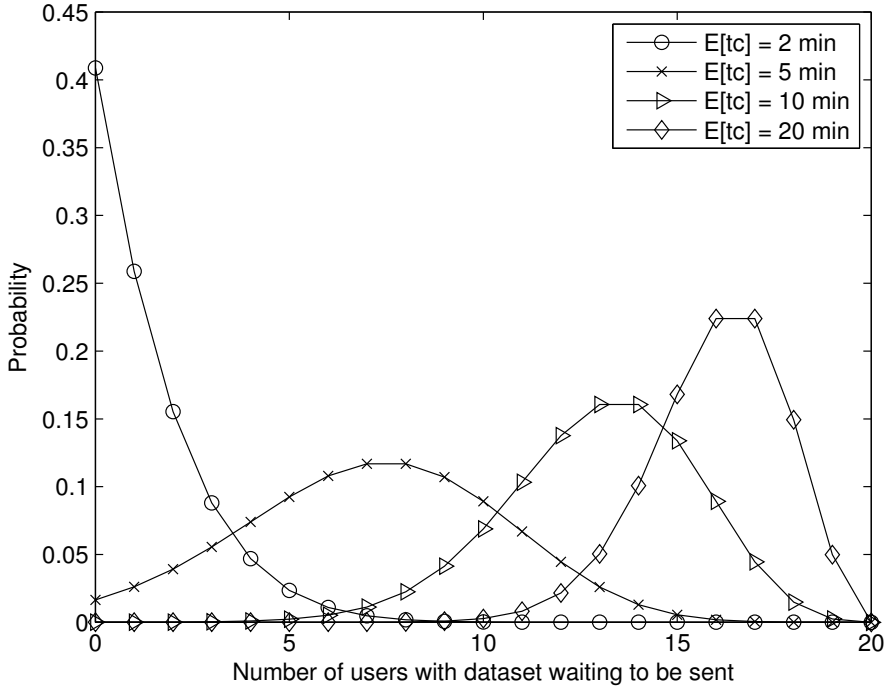


Figure 3-4: Probability density of number of users with a dataset waiting to be sent for the "multiple users - local computations" case

3-4 One user, computations remotely using one cloud

In this section, we will introduce the cloud as part of the system. This complicates the system in a certain way, as it will automatically add the network as a parameter. Networks and storage are well known to create bottlenecks in computation systems.

The system that we have focused on in this subsection can be seen schematically in Figure 3-5. We can see there all the different elements that the system consists of. One user that utilizes a datacenter to run services, one datacenter and a single cloud that is used as an IaaS platform. Finally, there is a network connection between the two computational spots.

3-4-1 Formulas

By introducing the cloud, more parameters make their appearance in the equations that compute the total response time. Some of these parameters have to do with the network and other with the cloud itself.

The data needs to traverse the network in order to reach the cloud. The network speed that is available affects the time the data spend in total on the network link.

Before the data communication starts, several virtual machines need to be launched in the cloud. This usually takes some time, unless they are already running. Finally, the computations themselves need to take place.

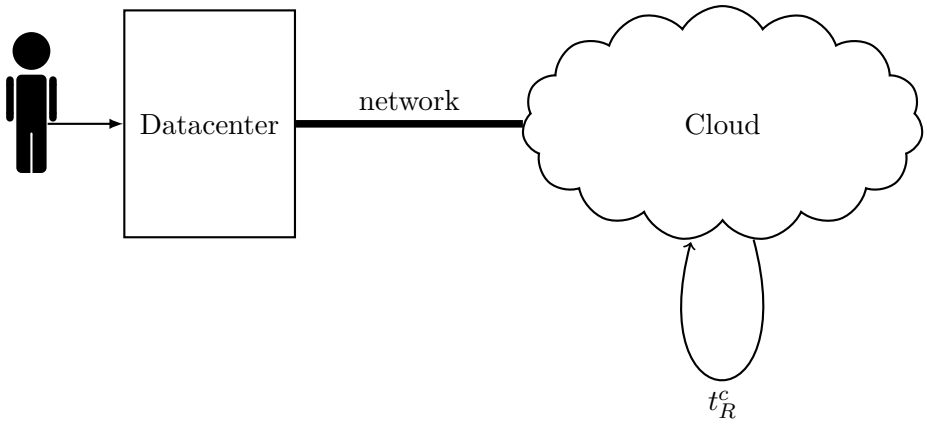


Figure 3-5: One user, remote computations using one cloud.

The following formulas apply for the total response time for the system described above.

$$t_R = t_R^c \times I \times (\frac{k}{v}) + \frac{n}{\mu} + S(v) \tag{3-10}$$

$$t_R = t_R^c \times I \times (\frac{k}{v})^2 + \frac{n}{\mu} + S(v) \tag{3-11}$$

$$t_R = t_R^c \times I \times (\frac{k}{v})^3 + \frac{n}{\mu} + S(v) \tag{3-12}$$

where t_R^c corresponds to the computation per dataset line in every Virtual Machine(s) (VM(s)) on the cloud, k corresponds to the dataset size in terms of elements and n to the dataset size in terms of bytes. The interference from the cloud activity is denoted with I, v is the number of VM(s) to be deployed on the cloud, μ is the network speed that connects the datacenter to the cloud and $S(v)$ is a time setup function[3] for the virtual machines. These equations refer respectively to $O(k)$, $O(k^2)$ and $O(k^3)$ classes of algorithms for the given input of data n which corresponds to a given amount of k elements.

3-5 One User, computations remotely using multiple clouds

There are even more ways to approach the system. By utilizing multiple clouds we enhance the system's performance due to various factors. First of all, with multiple cloud options, it is straightforward that there would be multiple network links connecting the datacenter with the different available computational spot. More links mean that data could be split and transferred in parallel, thus reducing the overall transfer time.

By splitting the data over various clouds, we consider that there is no need of any communication among the several computational spots, locally or remotely. That being said, we are ready to move on to the formulas that correspond to this particular scenario.

[3] $S(v) = \log_2 v \times c$

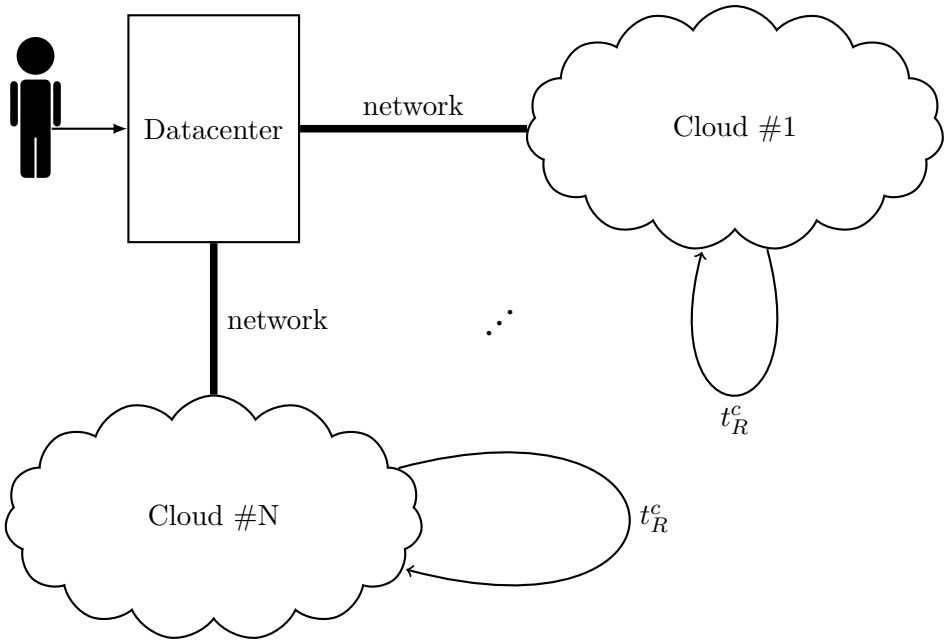


Figure 3-6: One user, remote computations using multiple clouds.

3-5-1 Formulas

The formulas are very similar to the previous scenario. Since the data is split among several clouds, we consider the total response time to be the maximum response time of each computational spot separately.

$$t_R = max(t_{Ri}^c \times I_i \times (\frac{k}{v_i}) + \frac{n}{\mu_i} + S_i(v_i)) \tag{3-13}$$

$$t_R = max(t_{Ri}^c \times I_i \times (\frac{k}{v_i})^2 + \frac{n}{\mu_i} + S_i(v_i)) \tag{3-14}$$

$$t_R = max(t_{Ri}^c \times I_i \times (\frac{k}{v_i})^3 + \frac{n}{\mu_i} + S_i(v_i)) \tag{3-15}$$

where t_{Ri}^c corresponds to the computation per dataset line in every VM(s) on cloud i, k corresponds to the dataset size in terms of elements and n to the dataset size in terms of bytes. The interference from the cloud activity is denoted with I_i, and it is different for each cloud, v is the number of VM(s) to be deployed on cloud i, μ_i is the network speed that connects the datacenter to the specific cloud and $S(v)$ is a time setup function[4] for the virtual machines. These equations refer respectively to $O(k)$, $O(k^2)$ and $O(k^3)$ classes of algorithms for the given input of data n which corresponds to a given amount of k elements.

[4] $S(v) = \log_2 v \times c_i$

3-6 Multiple users, computations remotely using one cloud

It is often the case that a single datacenter hosts multiple users that want to use a specific service offered. In this particular case, we consider that all the users want to run analytics of their own datasets, and the datacenter provides this service to all the users.

As a first approach to the multiple users case, we have considered here that the datasets are sent to a single cloud. This means that all the datasets compete over a single link that connects the datacenter to the cloud. Graphically, the system is represented in Figure 3-7.

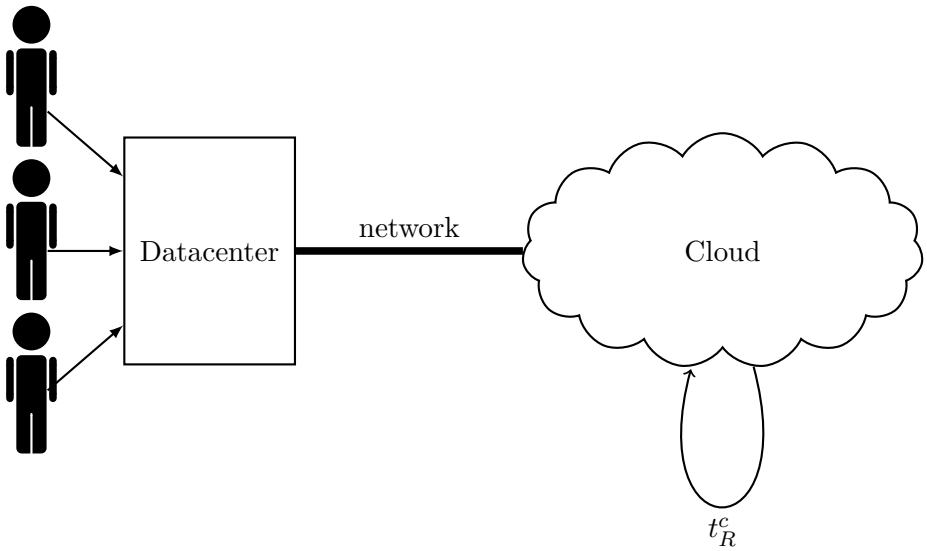


Figure 3-7: Multiple users, remote computations using one cloud.

The number of users in the network part of the system can be described using the Markov Chain that is shown in Figure 3-8. Every state of the Markov Chain shows the number of datasets in the system. State 1 means that the current dataset is being sent whereas states 2 and the remaining states to the right of 2 are the datasets waiting in the queue.

Figure 3-8: Markov Chain describing the network part of the multiple user scenario with one cloud.

3-6-1 Formulas

Since users are competing over a single link, a new parameter has been introduced in the equations that correspond to the waiting time in case someone is already transferring data. The modified equations for this system can be seen below.

$$t_R = t_R^c \times I \times (\frac{k}{v}) + \frac{n}{\mu} + S(v) + t_{wait} \tag{3-16}$$

$$t_R = t_R^c \times I \times (\frac{k}{v})^2 + \frac{n}{\mu} + S(v) + t_{wait} \tag{3-17}$$

$$t_R = t_R^c \times I \times (\frac{k}{v})^3 + \frac{n}{\mu} + S(v) + t_{wait} \tag{3-18}$$

where t_{wait} is the time each user waits in queue before their data is transmitted. These equations refer, just as before, respectively to $O(k)$, $O(k^2)$ and $O(k^3)$ classes of algorithms for the given input of data n which corresponds to a given amount of k elements.

For the waiting time, the concepts of the M/M/1 queuing systems are valid here. The normal behavior in a realistic scenario would be that the dataset arrivals would occur in a steady manner (for instance, once an hour at a particular time stamp selected by the individual user). In order to provide a mathematical basis, we have considered here that the arrival rate of datasets follow the Poisson distribution and that the service rate is exponentially distributed. The reason for this is that we cannot provide specific arrival times for any given scenario. Our assumption holds as, we are considering that the average rate of the Poisson distribution is the steady inter-arrival time selected by each user and we are assuming that its event occurrence is independent of the time since the last event. With this stochastic approach we provide a mathematical model that is widely used in queuing systems and thus is generally accepted. From [17], we know that:

$$E[w_{M/M/1}] = E[T] - \frac{1}{\mu} = \frac{\rho}{\mu(1-\rho)} \tag{3-19}$$

thus,

$$t_{wait} = \frac{\frac{\lambda_{total}}{\mu_{total}}}{\mu_{total} - \lambda_{total}} \text{ (hrs)} \tag{3-20}$$

and

$$\lambda_{total} = \sum_{1}^{n_{total}} \lambda_i \tag{3-21}$$

$$\mu_{total} = \frac{T}{\frac{n_{total}}{\lambda_{total}}} \tag{3-22}$$

where T is the available throughput (per hour) of the link under consideration.

3-7 Multiple users, computations remotely using multiple clouds

The last and most complicated case is a system consisting of multiple users in the datacenter and multiple clouds operating as computational spots. The system is graphically represented in Figure 3-9.

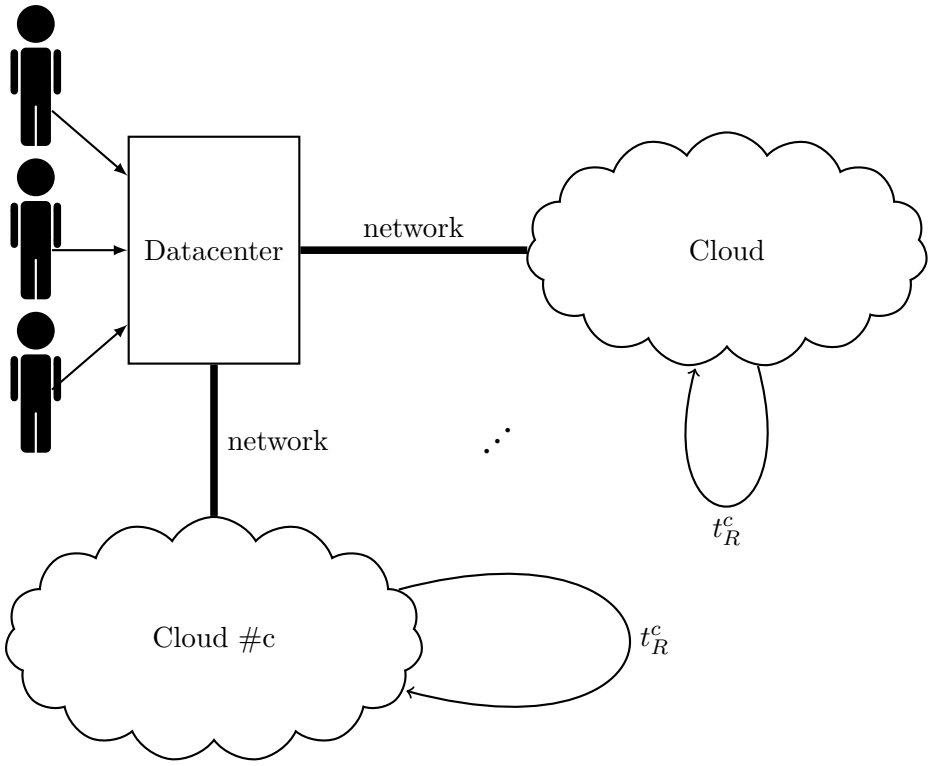


Figure 3-9: Multiple users, remote computations using multiple clouds.

3-7-1 Identical link rates

The network part of this system can be described using the M/M/c queuing model. Users are served in order of arrival. The state of the network is completely characterized by the number of users in the system. The system can be described using the Markov Chain shown in Figure 3-10.

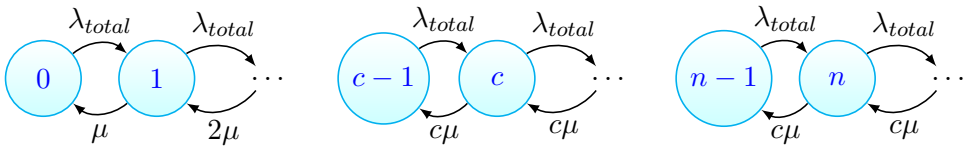


Figure 3-10: Markov Chain describing the network part of the multiple user scenario with multiple clouds and identical link rates.

We can derive the equilibrium equations for the probabilities p_n from the Markov Chain shown in Figure 3-10. According to [18], we can get simple equations by equating the flow out of and into the set of states $0, 1, \ldots, n-1$. This amounts to equating the flow between the two neighboring states $n-1$ and n yielding

$$\lambda p_{n-1} = \min(n, c)\mu p_n, \;\; n = 1, 2, \ldots \tag{3-23}$$

From the above equation we get

$$p_n = \frac{(cp)^n}{n!} o_0, \ n = 0, \ldots, c \tag{3-24}$$

$$p_{c+n} = \rho^n p_c = \rho^n \frac{(cp)^c}{c!} p_0, \ n = 0, 1, 2, \ldots \text{ and } \rho = \frac{\lambda}{c\mu} \tag{3-25}$$

Furthermore, from [18] we know that

$$E[W] = \Pi_W \frac{1}{1-\rho} \frac{1}{c\mu} \tag{3-26}$$

where

$$\Pi_W = \frac{(c\rho)^c}{c!} \left((1-\rho) \sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!} + \frac{(c\rho)^c}{c!} \right)^{-1} \tag{3-27}$$

3-7-2 Different link rates

When the link rates are different, it is not possible to use the M/M/c queuing model to describe the system. It is not possible to come up with closed-form equations due to the great number of varying elements that the system consists of. An equation is said to be a closed-form solution if it solves a given problem in terms of functions and mathematical operations from a given generally accepted set[5]. For this reason we will approach the system using a two-cloud scenario with different link rates connecting the datacenter with each of the clouds.

The queuing model for two clouds can be seen in Figure 3-11. There, we can see that the arrival rate in every queue is multiplied by a factor l. This factor corresponds to the the *queue load* and in practice it corresponds to the percent of datasets that end up in each queue. We consider here that this procedure is totally random.

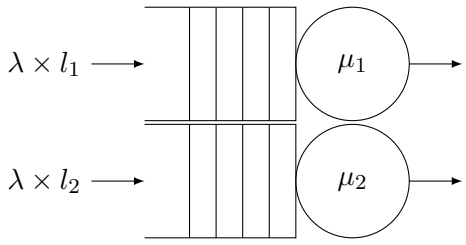


Figure 3-11: System model of multiple user scenario with multiple clouds and different link rates.

Using the results from the previous subsection, t_{wait} can be calculated using

[5]Weisstein, Eric W. "Closed-Form Solution." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Closed-FormSolution.html

$$t_{wait-Q1} = \frac{\frac{\lambda_{total} \times l_1}{\mu_{total-Q1}}}{\mu_{total-Q1} - \lambda_{total} \times l_1} \text{ (hrs)} \tag{3-28}$$

$$t_{wait-Q2} = \frac{\frac{\lambda_{total} \times l_2}{\mu_{total-Q2}}}{\mu_{total-Q2} - \lambda_{total} \times l_2} \text{ (hrs)} \tag{3-29}$$

and for the average t_{wait} we have

$$E[t_{wait}] = t_{wait-Q1} \times l_1 + t_{wait-Q2} \times l_2 \tag{3-30}$$

3-7-3 Formulas

Since the *different link rates* approach can incorporate the *identical link rates*, we will consider the t_{wait} formulas of the *different link rates* scenario. Having this in mind, the formulas to compute the response time are similar to the ones used in the previous system setups and can be seen below.

$$t_R = t^c_{Ri} \times I_i \times (\frac{k}{v_i}) + \frac{n}{\mu_i} + S_i(v_i) + t_{wait} \tag{3-31}$$

$$t_R = t^c_{Ri} \times I_i \times (\frac{k}{v_i})^2 + \frac{n}{\mu_i} + S_i(v_i) + t_{wait} \tag{3-32}$$

$$t_R = t^c_{Ri} \times I_i \times (\frac{k}{v_i})^3 + \frac{n}{\mu_i} + S_i(v_i) + t_{wait} \tag{3-33}$$

where for multiple clouds we consider that the data is sent to any available cloud i.

3-8 Comparison using realistic scenarios

3-8-1 Single user

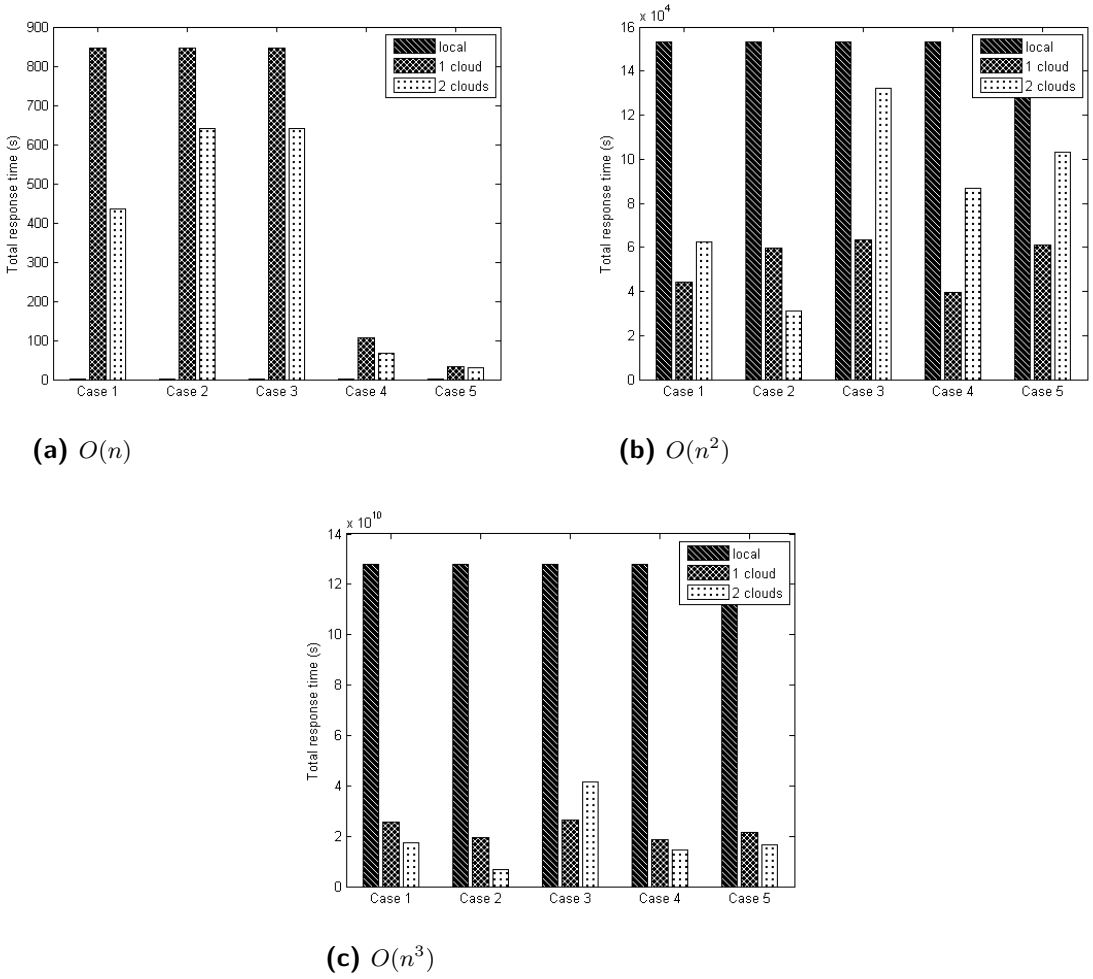
Let us first consider that there is one user/company utilizing a datacenter to provide a service to a set of customers/end-users. Let us consider the test cases shown in Table 3-5. To come up with the test cases as well as the specifications for the two clouds that are considered, we have considered a real experimental setup we have had available throughout this thesis work. This setup includes a centralized small server, access to an High Performance Computing (HPC) Cloud (Sara) using a private connection as well as the Internet and access to the Amazon EC2 service over the Internet.

For every case we have used the same type of dataset and the same algorithmic complexity $O(n)$. Furthermore in every cloud a total of 50 VMs are deployed for every job sent there. The results for the total response time can be seen in Figure 3-12a.

We can see from the plot that in every case, it is time efficient to run the analytics locally. The connection seems to play a very crucial role and the overhead that the network adds

Case Number	Dataset (Size - Lines)	Connection Throughput	Network Load Cloud "Sara"	Network Load Cloud 'Amazon'	Number of deployed VM(s)
1	1 GB - 20,029,940	10 Mbit	50%	50%	50
2		10 Mbit	75%	25%	
3		10 Mbit	25%	75%	
4		100 Mbit	50%	50%	
5		1 Gbit	50%	50%	

Table 3-5: Test cases to compare the single user scenario.



(a) $O(n)$

(b) $O(n^2)$

(c) $O(n^3)$

Figure 3-12: Comparison of total response time for a variety of algorithmic complexities spots and one user.

is enormous for computations of complexity $O(n)$. We can see that only when the network connectivity is changed to 100 Mbit that the total response time is close to the response time locally.

If we change the complexity to $O(n^2)$ or to $O(n^3)$ the situation is quite different as we can see in the plots in Figures 3-12b and 3-12c. We have considered here that all the other parameters have not changed.

Regarding the total response time, it is efficient to send the data to the cloud in all cases for complexities of $O(n^2)$ and $O(n^3)$. For case 1, we see from Figure 3-12b that it is more time efficient to use only one cloud. The reason for this is that the VM(s) from each of the two clouds are not identical, and to be exact, the VM(s) used in Amazon are a lot slower (see Table 3-2) since we have been considering the "t1-micro"[6].

3-8-2 Multiple users

Let us now consider two users in the datacenter. They both have datasets that they want to analyze and they have 3 options. Either use any available computation power in the datacenter or send the datasets to one or two available clouds. For the two clouds we have considered that the first one ('Sara') is a lot more expensive than the second one ('Amazon'). In this sense, we have considered here that we are able to deploy 200 VM(s) of type "micro" in Amazon at a lot less cost than 50 VM(s) of type small in Sara.

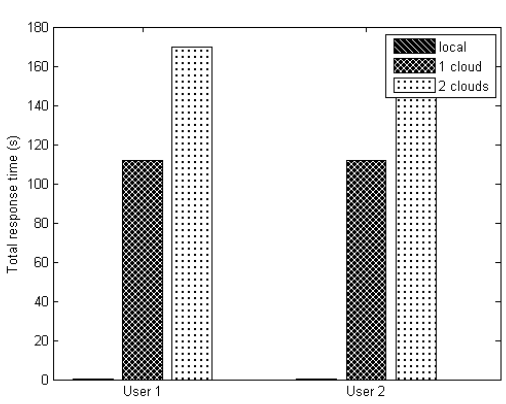
For the rest of the parameters, we will consider a single 1 Gbit dedicated link to Sara and 400 Mbps connection to Amazon. The rest of the parameters can be seen in Table 3-6.

User	Dataset (Size - Lines)	Connection Throughput ('Sara')	Connection Throughput ('Amazon')	Network Load ('Sara')	Network Load ('Amazon')	Number of deployed VM(s) ('Sara')	Number of deployed VM(s) ('Amazon')
1	1 GB - 20,029,940	1 Gbit	400 Mbit	50%	50%	50	200
2	1 GB - 27,843,152						

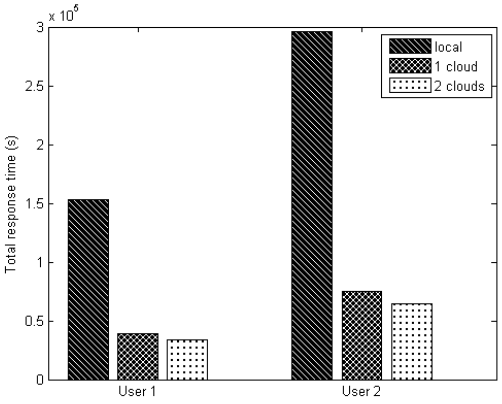
Table 3-6: Test cases to compare the multiple users scenario.

The results of the calculations can be seen in the plots in Figure 3-13. The results produce a twofold outcome. First of all, the network overhead is a bottleneck only when the calculations to be run are of time complexity $O(n)$. In every other case the bottleneck is the restricted available computation power locally. Moreover, having a second available connection is not that important compared to the more available computation power in the VM(s) of the first cloud. This can be identified when the time complexity is of class $O(n^3)$ where the more available computation power when utilizing both clouds reduces the total response time even more.

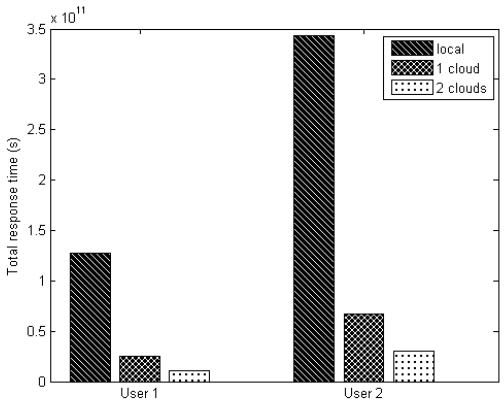
[6] http://aws.amazon.com/ec2/instance-types/ instance type for Amazon



(a) $O(n)$



(b) $O(n^2)$



(c) $O(n^3)$

Figure 3-13: Comparison of total response time for a variety of algorithmic complexities spots and multiple users.

Chapter 4

Simulation Approach

4-1 Introduction

An important step that needs to be taken, in order to understand the mathematical basis and to realize how the system would operate in a real-world system over time, is simulating the system by imitating the behaviour of a realistic system. To do so, we first need a solid simulation model that would depict the behavior of a real system as accurately as possible.

There are various techniques to run simulations of systems, not to mention existing tools that have been built for the whole purpose of running simulations. There are many, both open-source and commercial, network simulators like NetSim, Opnet, Omnet++ and NS2/NS3. The thing that all of these have in common, is that they focus on packet level simulation, which is of no interest to us here. In [19], *DeSiNe* has been presented as a flow-level network simulator which incorporates Quality of Service (QoS) routing and focuses on scalability. Because of the multilevel approach of this thesis, the simulations required a simple yet accurate model that would produce an as accurate as possible result. Our interest lies on the most important network metrics when considering flow-level control and large datasets. For this reason, in each of the following subsections, the simulators have been written from scratch, focusing on a simple abstract flow control level that would produce indicative results for point-to-point transfer of large datasets.

The simulation tool that has been used throughout this section is *Matlab*, as it provides a simple and efficient programming interface. Moreover, since it is a mathematical tool, it provides all the mathematical functions a simulator would require. All the simulations are event driven, which makes them demanding in time. On the other hand, a solid event driven simulation model would lead to accurate results. This work can be further supported as future work using more advanced techniques that consider more input parameters and or even advanced network typologies (see [19],[20]).

4-2 Scheduling multiple users over a single link

4-2-1 Simulation setup

Let us first begin with the simple case, where we only have one link on which to schedule the users' communication over the network. In order to realize how this would fit in a real world situation, let us consider that there are multiple Massively Multiplayer Online Games (MMOGs) companies hosted by the datacenter. Logfiles are created in real time for each of these games based on players activity. These logfiles need to be transferred to a storage service on the cloud in order to run some gaming analytics.

Popular MMOGs generate massive amounts of information; for instance, the database logging user actions for Everquest 2, a popular MMOGs, stores over 20 new terabytes (TB) of data per year [21]. Impressive as this number may seem, Everquest 2 is a MMOGs with 175 thousand active accounts[1], when, at the same time, there are other MMOGs with a total of 32 million active accounts.

Workload assumptions

Before getting into the range of possible policies, let us first make a number of simplifying assumptions about the processes running in the system, sometimes collectively called the workload.

The following assumptions have been used throughout this section concerning the processes, sometimes called jobs, that are active on the system:

1. The arrival of datasets for every user on the queue follows the Poisson distribution with mean arrival rate λ of one data-set per hour.
2. The service time of every data-set (transferring the data-set over the link) is linear to data-set input size, based on the throughput of the link.
3. For the computation part we have used numerical functions from the previous chapter (i.e. only the network part is simulated).

The first assumption stands from the fact that the players' activity throughout the day is not always the same. This means that in order to have a sufficient dataset size, the arrival rate of the complete dataset for each of these companies may vary. To show this stochastic nature we have selected the Poisson distribution, which stands for the activity of the user and hence for the dataset size. The remaining assumptions are straightforward from the system's set up that we have considered.

Link allocation policies

In this section we have simulated four different allocation policies for the scheduling of the datasets on the link. These are basic scheduling policies that are widely used in systems when

[1] MMOGChart.com

it comes to packet level link scheduling or job assignment in CPUs. Here we have used them to simulate the performance of the system when it comes to transferring big datasets over the network to a remote computation spot.

The allocation policies that we have simulated are:

- First In First Out (FIFO)
- Equal Shares (ES)
- Shortest Job First Preemptive (SJFP)
- Shortest Job First Non-Preemptive (SJF)

General notes

We consider a variety (k in total) of users (MMOGs companies) operating in the same datacenter. The size of the logging database of Everquest 2 will be used as a starting point to calculate the size of the logs generated for the other (fictional) MMOGs companies based on the total number of active users in a linear manner. The datacenter is connected to the storage service of the cloud using either a dedicated link or the internet. We will generally consider high available throughput for the dedicated link while using low and fluctuating throughput for the internet.

In Table4-1, one can see the users' data that were used throughout the simulations.

MMOG company	number of active users	logging database size per year
#1	175,000	20 TB
#2	300,000	34 TB
#3	400,000	45 TB
#4	800,000	91 TB
#5	1,000,000	114 TB
#6	2,000,000	228 TB

Table 4-1: Hypothetical users of a datacenter and their correspondent dataset sizes.

The average dataset size per hour for the various users in terms of file size and lines that we have used throughout the simulations can be seen in Table 4-2.

A variety of scenarios have been simulated to find the optimal network capacity for the given user setup. Using this information, one can optimize in terms of cost the network capacity that is required for the system to be stable. The same model can be used to find the optimal capacity for any users' scenarios.

More metrics like *link idle time*, *total response time* and *queue waiting time* can also be extracted. The first metric, along with the average queue size, shows how much extra network capacity is available than the one required. The second metric is important if we need to bound the total time that is required for a dataset to be analyzed from the moment it is created. Finally we should note at this point that all the numbers used throughout this chapter are in many cases as close as they can be to reality but still artificial in their nature.

MMOG company	file size	number of lines
#1	2.34 GB	13,763,640
#2	3.97 GB	79,528,455
#3	5.26 GB	176,720,940
#4	10.63 GB	328,875,826
#5	13.3 GB	195,464,950
#6	26.6 GB	287,678,603

Table 4-2: Average dataset size in terms of bytes and number of lines.

4-2-2 First in first out

The first allocation policy of interest is the "first in first out" queue, or abbreviated FIFO. The FIFO model, as the name implies, suggests that every element that enters the queue is serviced in order of arrival.

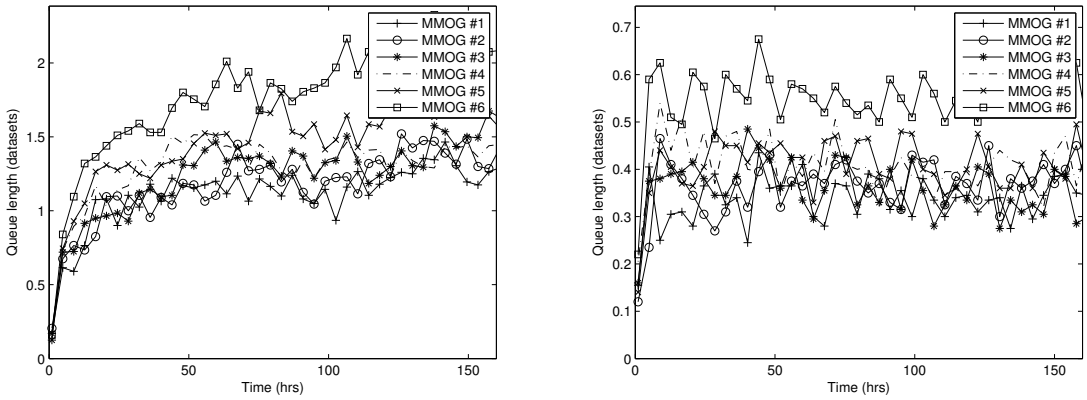
The simulation model that we used for the FIFO allocation policy can be seen in Figure B-1 (Appendix). To begin with, the model is initialized using the settings that we have predefined like throughput and total queue size. Other initialization settings include the arrival rate for the datasets of every user as well as the average dataset size for every user. The simulation as it was programmed follows the event driven scheme, which means that a target time must be set after which the simulation will end. Until that time is reached, in every timestamp the simulator checks whether a departure or an arrival of a dataset is taking place. If an arrival has just taken place, the queue size is increased. On the contrary, the queue size is reduced is a departure has just happened. Throughput this procedure, the datasets of the various users are put in a linked list and the first item is popped out every time a new departure is about to begin. For the total time that a dataset is transmitted over the network, no other dataset may start its own transmission.

Figure 4-1 shows average queue size throughout the simulation. Y-axis depicts the average number of datasets in the queue as the simulation time progresses (X-axis). These plots are the average of 200 repetitions of the same experiment. The stochastic nature of arrivals means that each of these executions was different from every other, which explains the fluctuation in the average queue size over time.

Results analysis

From the results in Figure 4-1 we can see that the queue is not stable for a total available throughput of less than 0.15 Gbit/s, thus the total number of waiting datasets would increase over time. This can be deducted from Figure 4-1a, as the datasets are gradually increasing over time. We observe that the increase in number of datasets in the queue is fairly small, which means that the queue can be stable with a small increase in the link capacity, but definately not stable for any value below 0.15 Gbit/s.

The fact that in Figure 4-1b all the lines overlap means that FIFO is a fair allocation policy that does not favor users. The dataset size has no effect on the queue size of the different users, at least for this experiment that all users have the same arrival rate. This would not have been the case for different arrival rate or in an unstable system.



(a) 0.15 Gbps - Average queue size per user

(b) 0.2 Gbps - Average queue size per user

Figure 4-1: Queue size throughout the FIFO queue simulation for various link capacities.

Finding the "golden ratio" for the value of throughput between a value that keeps the system stable and one that ensures that spikes in the arrival rate of datasets from the users will not be a cause for concern, is of utmost importance. This is the goal of this particular experiment and the reason that it is so important for configuring the system. Overcommiting resources is a well known practice, but how much can one overcommit without raising the costs too much? In our case, any number for the throughput value above 0.2 Gbps would have been sufficient for the system to be stable. The closer we are at 0.2 Gbps, though, would mean that possible spikes in user activity or future expansion, in terms of clients, may not be supported without upgrading the network link that connects the datacenter to the cloud.

4-2-3 Equal shares

The second allocation policy of interest is the "equal shares" system without a queue. The link is shared among the different users in parallel (i.e. every arriving dataset is transmitted right away).

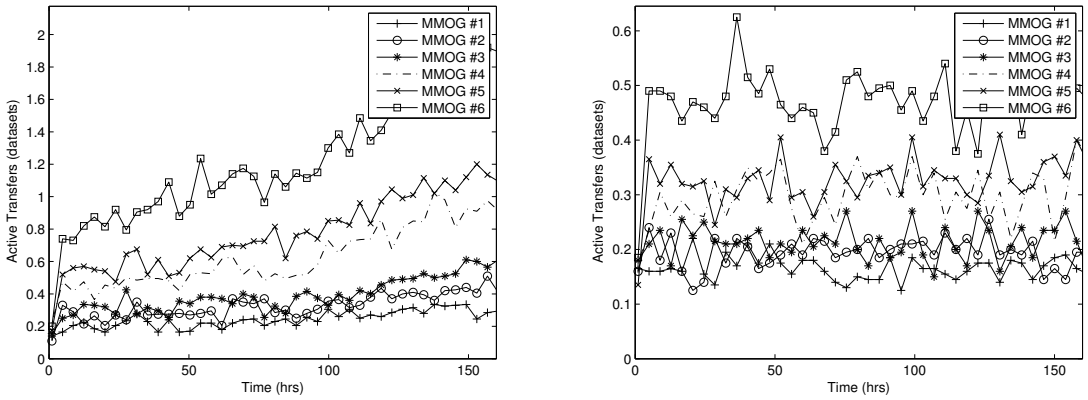
This allocation policy ensures that small datasets are transmitted fast and do not wait for a great amount of time in the queue in case big datasets have been placed in front of them. Although this could be achieved by using the SJF policy, the benefit from equal shares is that big datasets do not starve in the queue when there is a great number of small datasets arriving.

In this simulation model we can have unlimited simultaneous transfers, which means that we have to keep track of every dataset being sent. When there is a new dataset arriving, it starts to be transmitted right away. The departure time of every dataset depends on whether there are more datasets already being sent. If there are, then the departure time of those datasets needs to be updated as well, reason being that the throughput will be reduced from the newly arrived dataset.

The simulation model for the equal shares link allocation policy can be viewed in Figure B-2 (Appendix). The model is initialized in terms of throughput for the link that will be used for

the purpose of transferring datasets across the network, arrival rate for the datasets of the different users and average datasets size per user. The simulation takes place in a predefined amount of time. Until this target time has been reached, dataset arrivals are taking place. Upon arrival, the dataset starts to be transmitted right away. This is particularly complicated as every single dataset has to be tracked in terms of arrival and departure time. For this purpose an array has been used to keep track of all the active transfers. Moreover, the throughput of each dataset has to be modified after an arrival or a departure has taken place so that the aggregate throughput is constant. This has been programmed using a second array that has the same properties as the first one.

Figure 4-2 shows average number of datasets in the system throughout the simulation. Y-axis depicts the average number of datasets in the system as the simulation time progresses (X-axis). These plots are the average of repeated repetitions of the same experiment. The stochastic nature of arrivals means that each of these executions was different from every other, which explains the fluctuation in the average number of datasets in the system over time.



(a) 0.15 Gbps - Average queue size per user

(b) 0.2 Gbps - Average queue size per user

Figure 4-2: Active transfers throughout the ES queue simulation for various link capacities.

Results analysis

In Figure 4-2a we can see that the system is not stable, thus the total number of sending datasets increases over time. We observe that the increase in number of datasets in the system is fairly small, which means that the system can be stable with a small increase in the link capacity.

Figure 4-2b depicts a stable queue progression over time. We observe that the system is stable, hence the average number of datasets is stable over time. The same remarks as in the FIFO case are valid here as well. Same as before, the queue size of the different users is almost identical. In this case, any value for the throughput above 0.2 Gbps would have been sufficient for the system to be stable. The closer we are at 0.2 Gbps, though, would mean that possible spikes in user activity or future expansion in terms of clients may not

be supported without upgrading the network link that connects the datacenter to the cloud. Comparing this experiment with the FIFO case, it would seem that is is safer to keep the throughput closer to 0.2 Gbps. The reason is that the system manages to keep fewer total active transfers than the number of queued items for the FIFO queue for the same value of the link throughput.

There are certain drawbacks when considering the equal shares system. It is often the case that the user data consists of sensitive information, which the company may want to transfer over the network using an encrypted data stream. On the fly encryption using a protocol that supports it may lead to excessive CPU usage which, in turn, may lead to increased overhead if there is not sufficient CPU power for all the different parallel connections between the datacenter and the Virtual Machine(s) (VM(s)) deployed in the cloud.

4-2-4 Shortest job first - preemptive

The third allocation policy of interest is a queue that prioritizes the shortest jobs. Furthermore, in this subsection we are interested in the preemptive version of this allocation policy, which means that short jobs will put longer jobs on hold when they arrive.

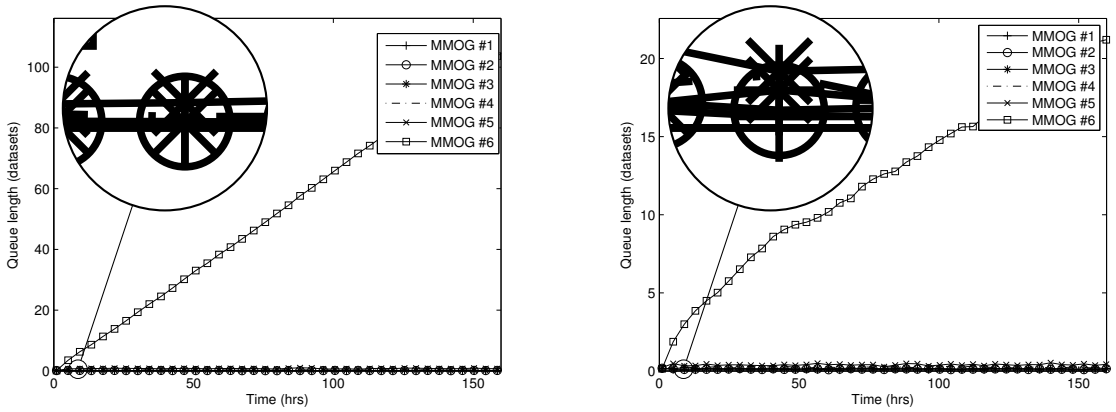
The simulation model that we used for the SJFP allocation policy can be seen in Figure B-3 (Appendix). At first, the various initializations take place, including the throughput of the link, the arrival rate of the users and the average dataset size for every user. For as long as the simulation takes place datasets arrive using the Poisson arrival distribution with an intensity that has been set above. When an arrival takes place, if there is another dataset already being transferred and its (initial) size is larger than the newly arrived one, then this dataset is put back in the queue and the new dataset starts to be transmitted. Datasets with size larger than the one transmitted are placed by default in the queue. In order to keep track of the datasets in the queue and the one actively transferred, a 2-dimensional array has been used to keep track of the multiple datasets from different users that are possibly waiting to be transmitted.

Figure 4-3 shows the average queue size throughout the simulation. Y-axis depicts the average number of datasets in the queue as the simulation time progresses (X-axis). These plots are the average of repeated repetitions of the same experiment. The stochastic nature of arrivals means that each of these executions was different from every other, which explains the fluctuation in the average queue size over time.

Results analysis

In Figure 4-3 we can see that the queue is not stable, thus the total number of waiting datasets increases over time. The previous statement is true for the sixth user, the user with the largest dataset. In either situations this leads the total number of datasets in the system to increase as well. Small datasets are always prioritized on arrival leading the largest dataset in the system to starve and hardly ever manage to traverse over the network. Even for 0.2 Gbit/s, a throughput that would keep the system stable for the FIFO and the ES allocation policy, the system fails to be stable.

Prioritizing small datasets is generally a problematic approach. It is noticeable that the system is hardly stable. Although small datasets are sent over the network quickly and



(a) 0.15 Gbps - Average queue size per user

(b) 0.2 Gbps - Average queue size per user

Figure 4-3: Queue size throughout the SJFP queue simulation for various link capacities.

efficiently, when these datasets are combined with very large ones, the latter are starving which is a highly undesired effect. For this reason the preemptive version of SJF should not be utilized when we are dealing with big datasets, that in many cases are required to be prioritized over smaller ones, since larger datasets would mean in general bigger companies with more demands in terms of service level agreement with the datacenter.

4-2-5 Shortest job first

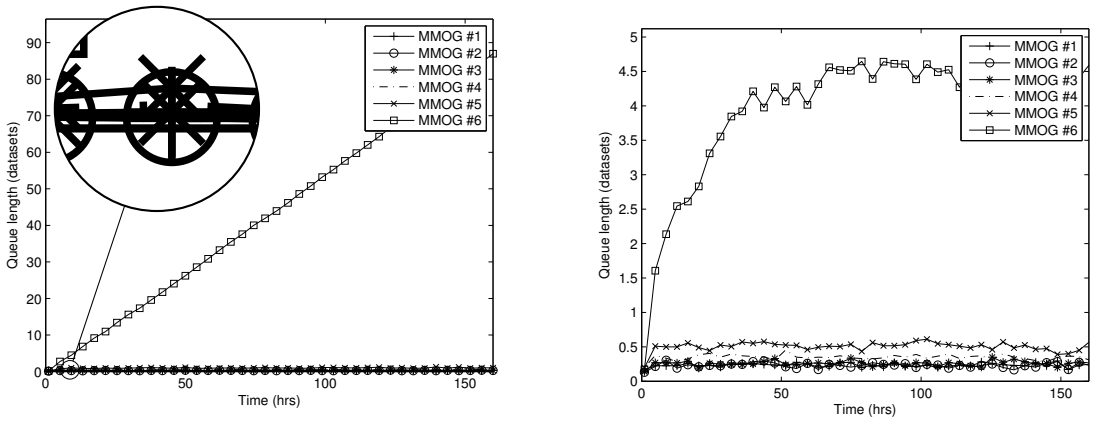
The fourth allocation policy of interest is a queue that prioritizes the shortest jobs without preemption. The simulation model that we used for the SJFP allocation policy can be seen in Figure B-4 (Appendix).

Like previously, the various initializations take place, including the throughput, the arrival rate of the users, the average dataset size for every user and the Poisson arrival distribution rate. Newly arriving datasets are always placed in the queue which, in turn, has been modeled using a 2-dimensional array to keep track of the available dataset sizes along with the total number of datasets per user.

Figure 4-4 shows average queue size throughout the simulation. Y-axis depicts the average number of datasets in the queue as the simulation time progresses (X-axis). These plots are the average of repeated repetitions of the same experiment. The stochastic nature of arrivals means that each of these executions was different from every other, which explains the fluctuation in the average queue size over time.

Results analysis

In Figure 4-4a we can see that the queue is not stable, thus the total number of waiting datasets increases over time. This is also the case in Figure 4-4b, although the effect here



(a) 0.15 Gbps - Average queue size per user

(b) 0.2 Gbps - Average queue size per user

Figure 4-4: Queue size throughout the SJF queue simulation for various link capacities.

is minor and we can consider that the system is very close to stable with a large number of waiting datasets, at least for the largest dataset.

For a throughput value larger than 0.2 Gbit/s the system would have been stable. Comparing this to the FIFO and the ES allocation policy would mean that a higher total throughput is required for the system to function properly.

Compared to the preemptive case of the same allocation policy we observer that the total system behavior here is better, as it was expected. The effect of starvation is not as intense as in the preemptive version of the SJF allocation policy. This means that it can be used for allocating big sets of data over the network. When overcommiting network resources, SJF could be a solid option if one is interested in prioritizing smaller datasets without leading larger datasets to starvation.

4-2-6 Other system metrics

Idle link

Throughout the simulations, the link has been idle (i.e. there were no active transfers) for smaller or larger periods of time. The amount of idle time can be seen in Figure 4-5.

We observe that the link was more idle the larger the throughput was. By carefully inspecting the graph and by combining the results from the previous subsections one can find the optimal link capacity for the given number of users in the simulations.

It is important to mention why there is so much idle time even for small link capacity figures. The reason is that the dataset arrivals followed the same distribution. Starting from time zero, all the datasets arrivals followed the Poisson distribution with mean inter-arrival time of 1 hour. Because of this, datasets where arriving in many cases close to each other leading to an increase in the queue size. Making this inter-arrival time differ among the several users could lead to better link management and optimization of the whole process.

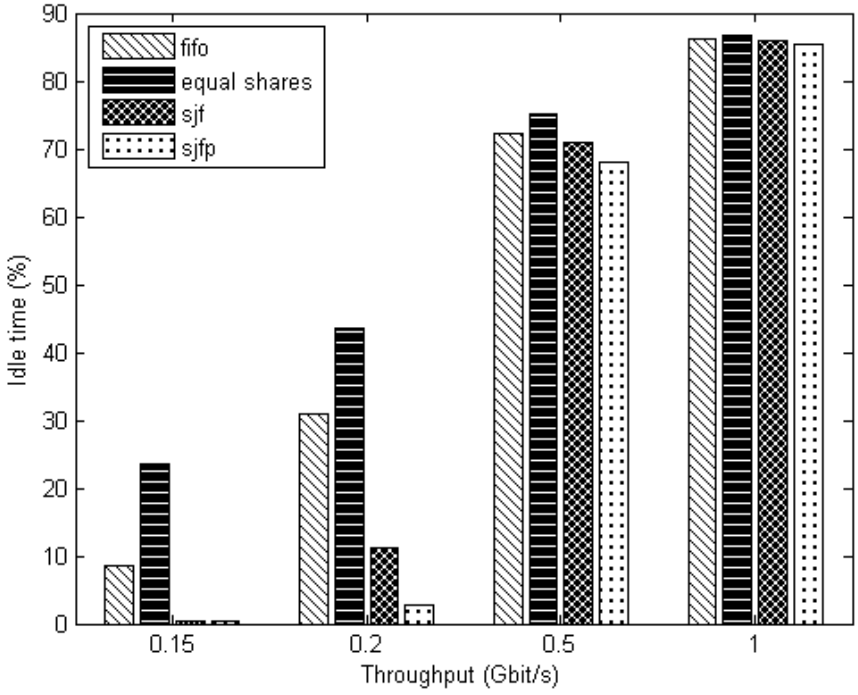


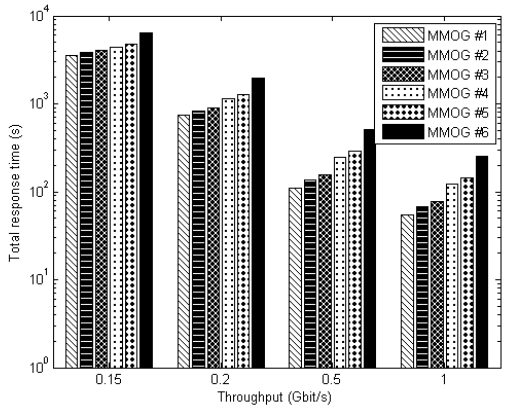
Figure 4-5: Idle Time of the link for different allocation policies and link speed.

Total response time

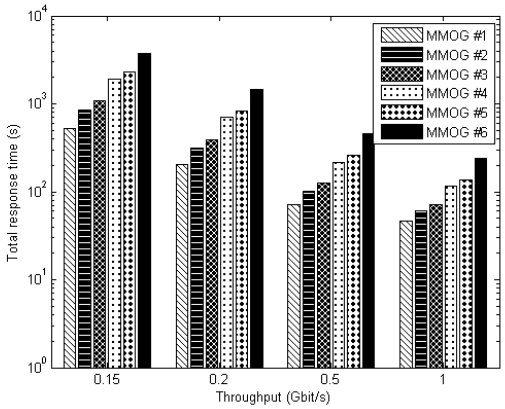
The metric that interests us the most throughout this thesis is the *total response time*. For each of the previous simulation tests, we have presented in Figure 4-6 the total response time results.

As it was anticipated, the higher the available throughput, the better is the total response time we have got. It is important to notice the difference in the response time of the various users (test cases) for different link allocation policies. There is a huge difference observed when using SJF or even SJFP which is a result of the starvation for users with bigger datasets, as we have discussed earlier.

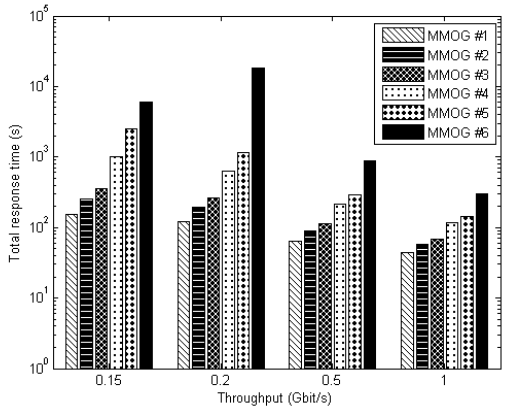
By carefully observing 'MMOG #6' when using any of the shortest job policies, one should notice the lower response time for throughput of 0.15 Gbit/s compared to the case with 0.2 Gbit/s. Since we have been using an event driven simulator that we deployed for a certain period of time, there were still datasets waiting in the queue when the simulation was stopped. For this reason, these datasets do not count towards the result of the total response time, since the result is not known by the time the simulation ends, which is the reason for this particular behavior shown in the graphs.



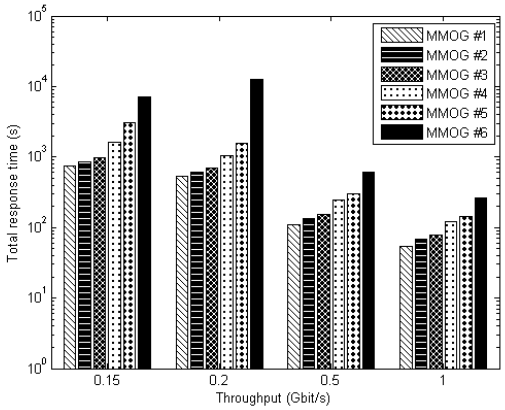
(a) FIFO



(b) ES



(c) SJFP



(d) SJF

Figure 4-6: Total response time results.

4-3 Scheduling multiple users over multiple links

4-3-1 System setup

Let us consider now a more complicated case, where the communication of multiple users to various computation spots (i.e. IaaS clouds) has to be scheduled efficiently. Studying the optimal allocation policy for multi-queuing server systems is not an easy task, as closed-form equations for this scenario are very difficult to come up with. There are various approaches in the literature to the resource allocation problems of multi-queuing systems.

In [22], the authors have studied a system in which two types of jobs are served by a single pool of resources. They have associated priorities, based on an ageing factor that grows proportionally with the waiting time and gave an analytical model for computing the expected waiting. This heuristic, was first analyzed by Kleinrock in [23] and [24]. [25] applies more complex heuristics in a multi-skilled queueing system with as performance metric the tail

probabilities of the waiting time. The authors compared the number of jobs in each facility that actually have been served to the number that, nominally, should have been served under a long-run average allocation scheme. The "further behind" the actual number of services, the higher the resulting priority. [26] and [27] study a variant in which only one queue has a QoS-constraint. They used Markov decision processes and Linear Programming to obtain optimal or nearly-optimal control strategies. [28] simplifies the problem by using fixed, static priority policies using matrix-geometric methods. In [29] they have dynamically allocated the servers over the different computation spots such that the sojourn-time constraints are met at minimal costs. They modelled this problem as a Markov decision process and derived structural properties of the relative value function. Their findings only support identical servers with identical costs per server at each facility, and its facility is handled independently.

The system we are looking at is extremely dynamic. The reason for this is that all the components of the system are different. A decision needs to be made among different network links, in terms of throughput as well as all the other network metric and different VM(s) specifications, in terms of computation power, memory and available space. Finally, the pricing schemes may differ as well.

Despite the dynamic nature of our system, the one thing that we can consider here is that the system size is bound. Whether we take into account the users that needs to be served, the different network options or the total number of VM(s) we need to deploy, the numbers are bound to such a degree that we can actually decide on the optimal decision extremely fast. Although this may not be the case for extremely busy datacenters, we have focused on this simple solution and we leave the implementation of more complicated scheduling policies as future work.

Workload assumptions

Same as with the single link case, we make here a few assumptions for the workload of the system.

The same assumptions as before have been used throughout this section concerning the processes that are active on the system:

1. The arrival of datasets for every user in the system follows the Poisson distribution with mean arrival rate λ of one dataset per hour.
2. The service time of every dataset (transferring the dataset over the link) is linear to dataset input size, based on the throughput of the link that is selected.
3. For the computation part we have used numerical functions from the previous chapter (i.e. only the network part is simulated).

Link scheduling policy

Based on all the previous facts, we are presenting here the scheduling policy that will be used to simulate the multiple queue system. We are talking about link scheduling policy as we have to decide on which link to select and then we will use link allocation policies separately

for each of the links. We have considered the cost and total response time metrics for the scheduling policy.

For our scheduling framework, the following rules have been selected. In the first step, a decision is taken on how many parallel connections are required over a certain link, to optimize the aggregate network attained throughput. Although the network capacity may be large, there are other aspects that may reduce the attained throughput, like limited physical drive read and write I/O speed or network limits per network connection. After finding the optimal minimum number of connections, which is translated to the minimum number of VM(s) to be deployed, the total number of VM(s) is calculated for which the total response time satisfies the bounds the user has selected for the total response time. In this second step, the total response time is not optimized, but the value for which the cost is minimum is selected. The final value for the total number of deployed VM(s), and hence the total simultaneous transfers, is the maximum of the previous two.

For the previous scheduling policy to work, a few system settings have to be given as an input to the system. These settings include the network throughput of every network link and the network throughput per connection to the target computation spot. The computation time per input line is also required, as it was computed in "Analytical approach" (Table 3-3). Based on the link characteristics, different link allocation policies per link may also be selected.

Simulation setup

In order to simulate a realistic system scenario, we have considered the following characteristics. A datacenter is connected to two IaaS clouds. The network communication is handled by two different network links. The first is a gigabit optical line and the second is an internet connection with data rate of 100 Mbit. A graphical representation of the system setup can be seen in Figure 4-7.

Based on real measurements taken by both of these IaaS Clouds, we are also considering the following restrictions:

- Sara HPC Cloud has, on average, 200 Mbit of I/O speed per VM(s).
- Amazon EC2 has a limit of 3 Mbit of throughput towards every micro VM(s).

Using the previous information, it is evident that we need at least 5 parallel connections to Sara HPC cloud to maximize the network throughput and at least 34 parallel connections to Amazon EC2. For each of the links, we have used both the FIFO and the ES allocation policy. SJF has not been used as it leads to starvation for the big datasets, which, as explained previously, is a truly undesired behavior.

We have considered that ten users are utilizing the datacenter and each user produces every hour datasets that require to be analyzed. We have considered $O(n)$, $O(n^2)$ and $O(n^3)$ complexities for the analytics. The dataset sizes each user produces follows the uniform distribution as shown in Table 4-3.

Based on the setup described above, we have run the simulation to find the queue size over time. We have also calculated the cost for every user regarding the VM(s) the scheduler

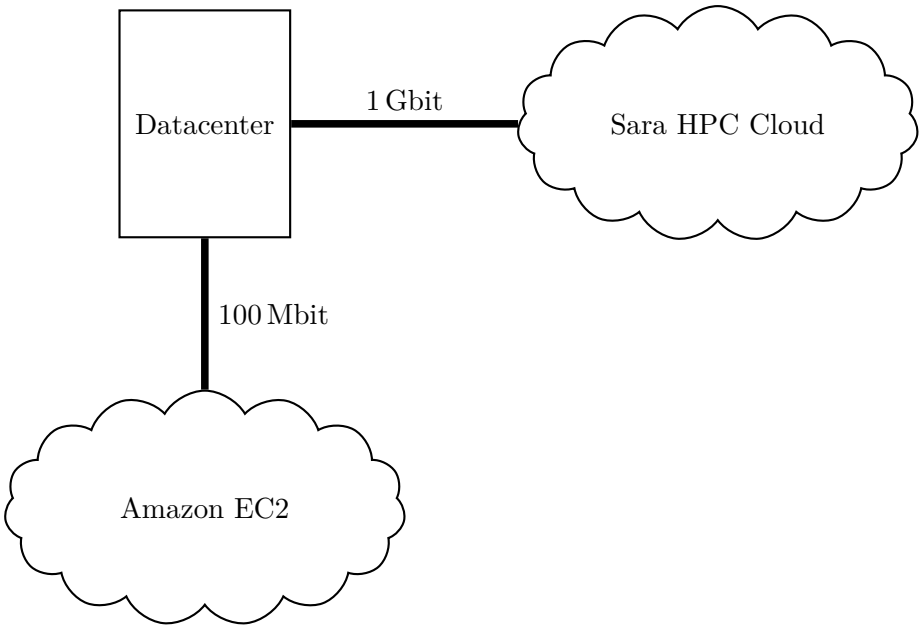


Figure 4-7: Simulating multiple users over multiple links.

User	Dataset size (GB)
#1	$\mathcal{U}(1, 10)$
#2	$\mathcal{U}(5, 10)$
#3	$\mathcal{U}(5, 25)$
#4	$\mathcal{U}(10, 30)$
#5	$\mathcal{U}(15, 30)$
#6	$\mathcal{U}(15, 45)$
#7	$\mathcal{U}(25, 45)$
#8	$\mathcal{U}(30, 50)$
#9	$\mathcal{U}(40, 60)$
#10	$\mathcal{U}(80, 100)$

Table 4-3: dataset size distribution for the users considered for the multiple users over multiple links simulation.

selects to keep the total response time into bounds. We consider that every user requires that the response time does not exceed 1 hour for the analytics of class $O(n)$ and 24 hours for the analytics of class $O(n^2)$.

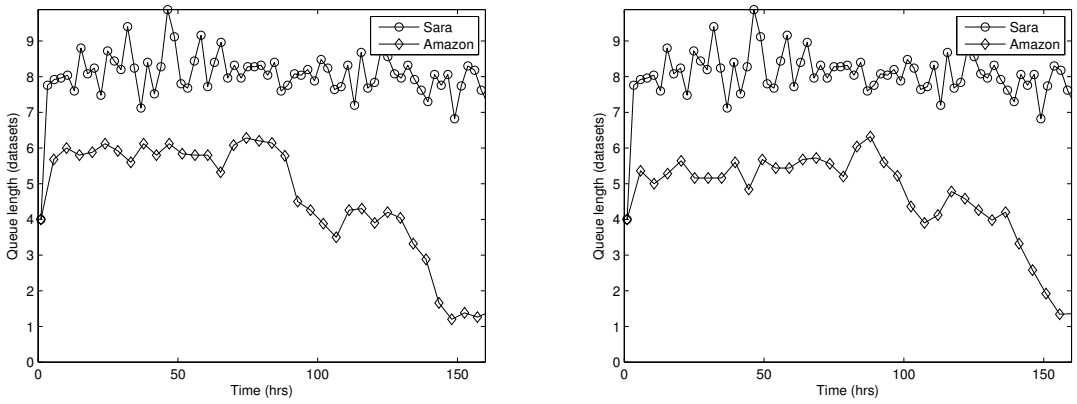
The scheduler selects Amazon for all the datasets with size smaller than 15 GB and Sara otherwise. In all the following cases, FIFO was selected as the link allocation policy for the Sara cloud. The reason is that Sara is selected for the very big datasets and we want this data to be transferred as fast as possible without any interference from other network activity. On the other hand, we have simulated two different allocation policies for the Internet link that connects the datacenter to Amazon cloud. The first one is FIFO, for which we have selected a minimum of 34 parallel connections to utilize all the available throughput (as

explained above) and ES, for which the minimum connections per user is one. In the later case, multiple transfers from different users will make sure that we utilize all the available throughput.

Regarding the cost, we have considered the value of \$0.02 per VM(s) in Amazon and \$0.12 per VM(s) in Sara (both values are per hour of usage).

4-3-2 Simulations with analytics complexity $O(n)$

The result of how the queue size looks over time throughout the experiment can be seen in Figure 4-8. We have considered for the computation class of $O(n)$ that the time we require to get the results back is one hour from the moment the dataset has been created. The results are the average of a total of 200 repetitions.



(a) FIFO link policy for Amazon

(b) ES link policy for Amazon

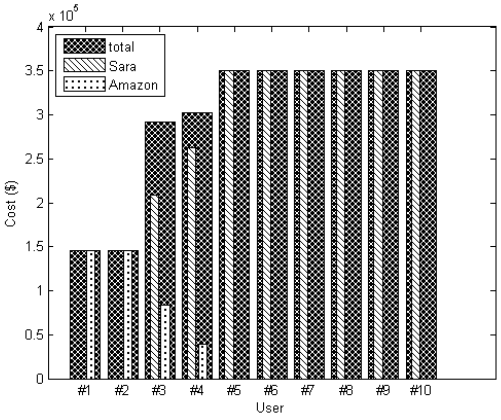
Figure 4-8: Queue size throughout the multiple clouds simulation for different link allocation policies ($O(n)$).

We can see that both queues (parallel transfers in the case of ES allocation policy) are stable over time. To be exact, both graphs look almost the same. What is of major interest is the VM(s) the scheduler selects in these two cases. The number of deployed VM(s), in terms of total cost, throughout the simulation time of 9 days can be seen in Figure 4-9.

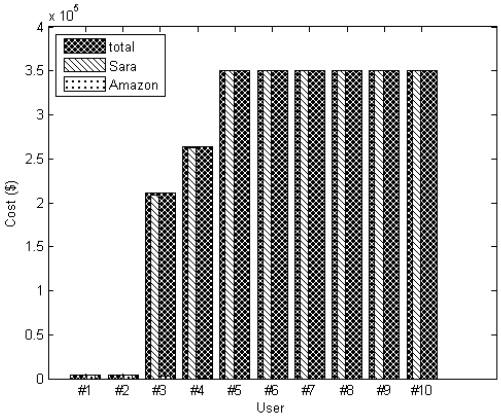
We can see that there is a major difference in the cost regarding the users that have smaller datasets. The reason is the fact that the scheduler is free to select any number for the deployed number of VM(s), which corresponds to lower overall cost.

4-3-3 Simulations with analytics complexity $O(n^2)$

Considering the analytics of complexity $O(n^2)$, almost nothing changes in the network part. Small differences are a result of the stochastic nature of the simulation. The same number of repetitions for the simulations has been used here as before. As we can see in Figure 4-10, the queue size over time is the same as before. We have considered for the computation class



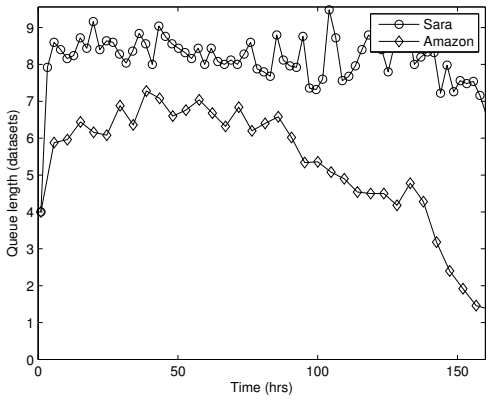
(a) FIFO link policy for Amazon



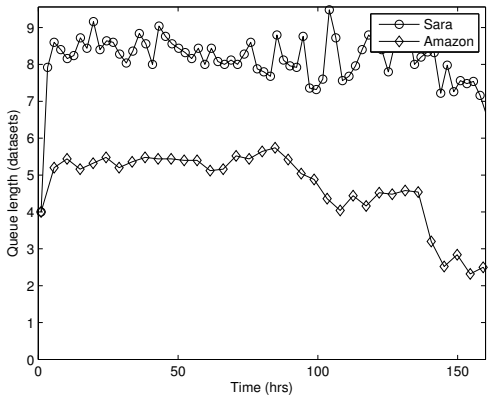
(b) ES link policy for Amazon

Figure 4-9: Cost for the multiple clouds simulation for different link allocation policies ($O(n)$).

of $O(n^2)$ that the time we require to get the results back is 24 hours from the moment the dataset has been created.



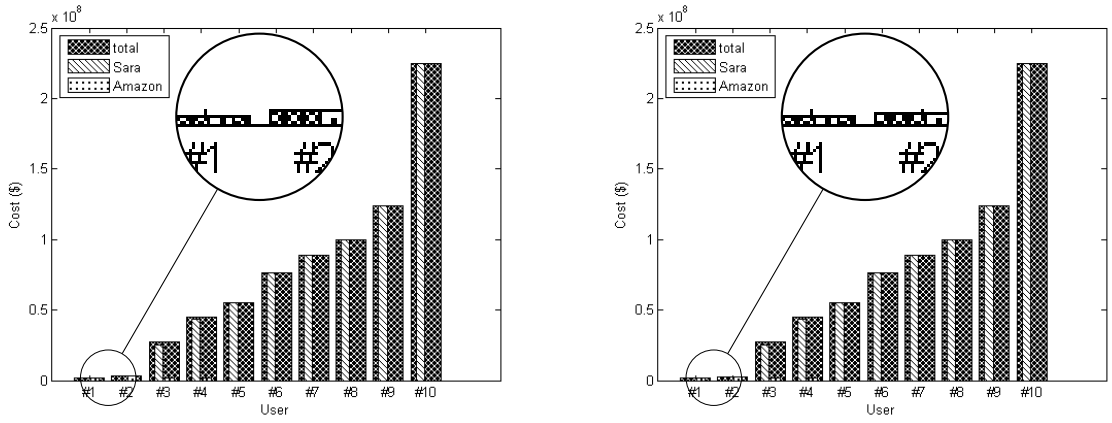
(a) FIFO link policy for Amazon



(b) ES link policy for Amazon

Figure 4-10: Queue size throughout the multiple clouds simulation for different link allocation policies ($O(n^2)$).

The cost on the other hand is now the same whether we have selected FIFO or ES for the link allocation policy in Amazon cloud, as we see from Figure 4-11. The reason is the great number of VM(s) that are required for the computations to finish within 24 hours. In both cases the scheduler selects the same great amount of VM(s) which makes the cost of the analytics procedure extremely expensive (millions of dollars for 9 days of analytics).



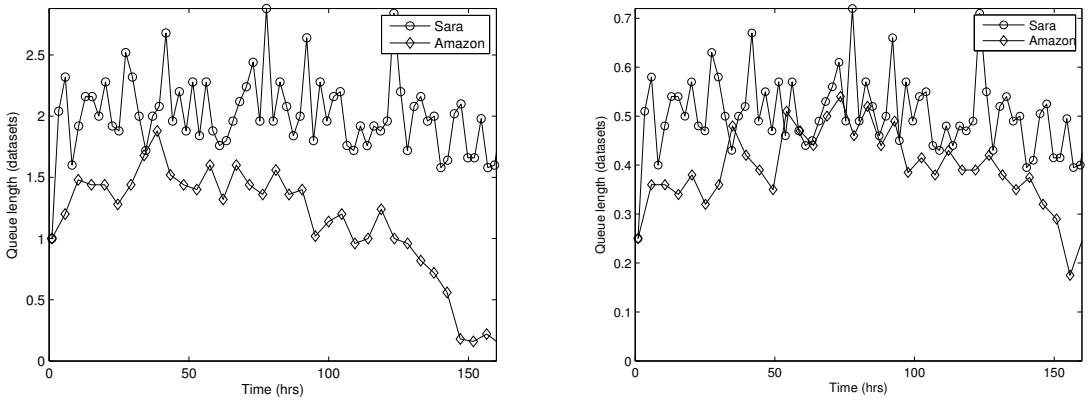
(a) FIFO link policy for Amazon

(b) ES link policy for Amazon

Figure 4-11: Cost for the multiple clouds simulation for different link allocation policies ($O(n^2)$).

4-3-4 Simulations with analytics complexity $O(n^3)$

When we consider the analytics of complexity $O(n^3)$, compared to the previous cases nothing changes in the network part. As we can see in Figure 4-12, the queue size over time is the same as before. There difference when using the ES allocation policy with Amazon can be explained from the stochastic nature along with the fewer repetitions.



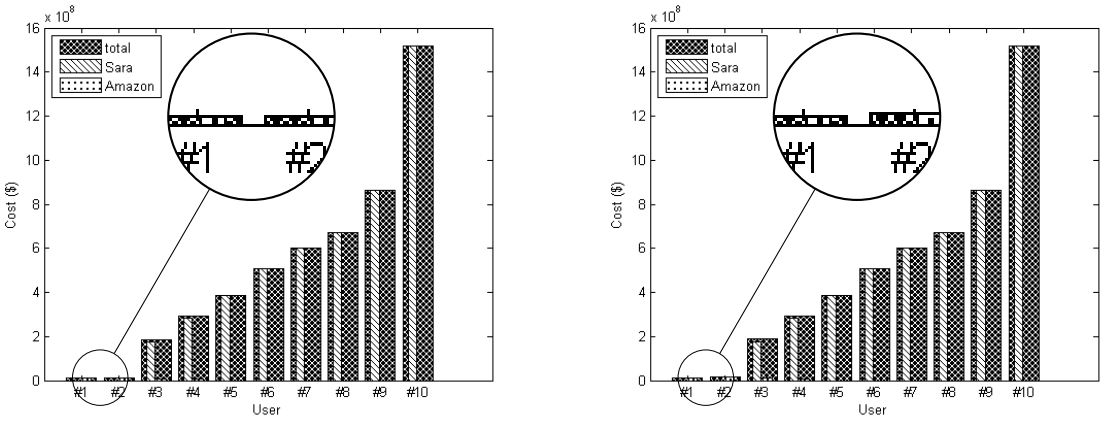
(a) FIFO link policy for Amazon

(b) ES link policy for Amazon

Figure 4-12: Queue size throughout the multiple clouds simulation for different link allocation policies ($O(n^3)$).

We have considered for the computation class of $O(n)$ that the time we require to get the results back is one month from the moment the dataset has been created. A total of 50 repetitions has been used to get the average that is depicted in the results. The reason that

we chose 50 repetition is because for this class of algorithm the simulation required a lot more time for the results to converge.



(a) FIFO link policy for Amazon

(b) ES link policy for Amazon

Figure 4-13: Cost for the multiple clouds simulation for different link allocation policies ($O(n^3)$).

The cost, like in the case of analytics of complexity $O(n^2)$, is again the same whether we have selected FIFO or ES for the link allocation policy in Amazon cloud, as it can be observed in Figure 4-13. The reason is again the great number of VM(s) that are required for the computations to finish within 24 hours. In both cases the scheduler selects the same great amount of VM(s) which makes the cost of the analytics procedure extremely expensive like in the previous case (millions of dollars for 9 days of analytics).

4-3-5 Conclusion

The previous results are very interesting and at the same time they show the extreme computational power that is required to handle time complexities of $O(n^2)$ and above. These results can be considered as the learning process of a scheduler that will fast and efficiently select the number of VM(s) to optimize the whole process.

On the downside, the enormous cost may drive small companies away of running so complex analytics with the produced datasets. The results show the need to optimize the analytics procedure in order to get the most out of the available VM(s). It is often the case that analytics algorithms do not utilize the maximum of the potentials of the CPU power available in a single VM(s).

Another approach could be to use the same VM(s) for analytics of class $O(n)$ along with other computations. It has been shown that even for very big sets of data, when the time complexity is $O(n)$ the computations are extremely fast, maybe even faster than what the user requires. Combining those analytics with analytics of class $O(n^2)$ or even $O(n^3)$ may lead to cost reduction. We leave this as future work.

Chapter 5

Real-world Experiment - Scheduler

5-1 Introduction

5-1-1 General information

In order to put the whole concept in practice and check the accuracy of the mathematical and simulation models, we set up a real world experiment. The experiment included building an experimental scheduling framework for transferring big datasets and running analytics on one or more Infrastructure as a Service (IaaS) clouds.

A complete overview of how the scheduling framework was built and how all the different functions operate will follow. All the elements as well as the structure of the scheduler and the system setup are included.

5-1-2 Technical information

Based on widely used Application Programming Interfaces (APIs) across the network and cloud programming community, including the *Amazon Web Services SDK*[1], for communication with the Amazon Cloud, an SSHv2 library for Java[2] and the *XML-RPC API*[3] for *OpenNebula*[4], we have developed our framework using Java. The framework was initially built to support two clouds:

1. Sara HPC Cloud[5]

[1] http://aws.amazon.com/sdkforjava/

[2] *sshj*, https://github.com/shikhar/sshj

[3] http://archives.opennebula.org/documentation:archives:rel3.2:api

[4] OpenNebula is an open-source cloud computing toolkit for managing heterogeneous distributed data center infrastructures.

[5] https://www.cloud.sara.nl/

2. Amazon Elastic Compute Cloud (Amazon EC2)[6]

Although the framework was fully built and supports both of these services, the experiments themselves were only run using Sara HPC Cloud. The reason for this was that the Free Tier that was used from Amazon to set up the framework was not sufficient to conduct experiments on big sets of data, due to enormous constraints in terms of free space and inbound traffic to the cloud. Most importantly, Sara's HPC Cloud proved to be more than enough.

Two connectivity options were available between the server used to deploy the experimental framework and the cloud. The first was a dedicated optical line with a nominal throughput of 1 Gbps that was deployed for the whole purpose of running these experiments and thus, there was no other traffic across this path. The second was the available internet connectivity. Since both of these infrastructures are hosted in the Netherlands and they both have multi-gigabit connections to the internet, the throughput was bounded by the 1 Gbit Ethernet connections that connected internally the subsystems that were used.

5-1-3 Benchmarking the private connection to the cloud

In order to get the best results out of our system setup, we had to tweak the network parameters of the operating systems that we have been using throughout the experimental procedure. Network cards, when not properly configured, may harm the overall performance of the system. They are usually set up for optimal performance around the 100 Mbit/s range of throughput speed.

Based on the results provided in [30], we have modified the TCP connection settings to support higher total throughput in the system. The setting we have selected are summarized in Table 5-1, where all values are in bytes.

Tweaked TCP Settings	
Max OS receive buffer size	8,388,608
Max OS send buffer size	8,388,608
Default OS receive buffer size	65,536
Default OS send buffer size	65,536
TCP stack	8,388,608/8,388,608/8,388,608
Min/Default/Max TCP receive buffer	4,096/87,380/8,388,608
Min/Default/Max TCP send buffer	4,096/65,536/8,388,608

Table 5-1: Modified TCP settings used in both the server and the cloud to optimize throughput (all values are in bytes).

Before deploying the experimental framework to run our tests, it was of paramount importance to benchmark the private connection we had available between our test server and the HPC Cloud. To do so we used *Iperf*[7], a well known network testing tool, to conduct throughput experiments explicitly from the server towards the cloud. We deployed the tool for one consecutive hour and kept track of the throughput in intervals of one minute. The results can be seen in Figure 5-1.

[6] http://aws.amazon.com/ec2/

[7] Iperf was developed by NLANR/DAST as a modern alternative for measuring maximum TCP and UDP

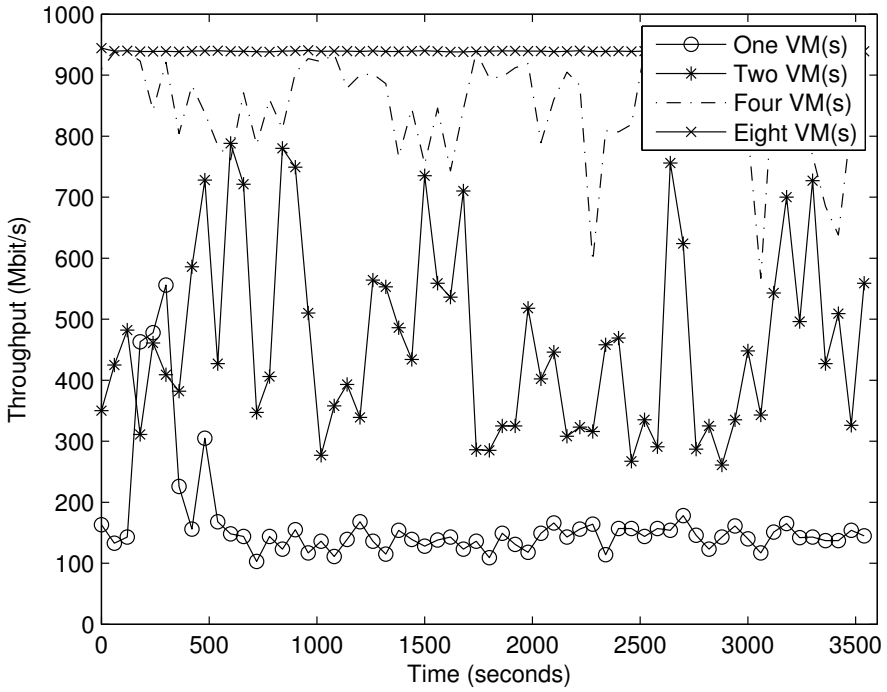


Figure 5-1: Throughput benchmark results of our system using iperf.

The results show that in order to optimize the aggregate throughput of the system, it is required to transmit data in parallel to multiple destination Virtual Machine(s) (VM(s)). During our experiments, transferring data to 8 different VM(s) led to a total of 950 Mbit/s of total aggregate throughput. For the remaining tests, not only the aggregate throughput is not maximum, but we also noticed an enormous throughput fluctuation during the experiments. Since the private line that we have been using was only utilized for our experiments, the reason for this fluctuation cannot be fully explained from our end of the network and could be a result of heavy network traffic in the cloud. Another reason could be network settings beyond our control or even the virtualization technology used in the cloud.

5-2 Single link experiments

5-2-1 System setup

For the single cloud case, the experimental framework was deployed on a server that had access to the HPC Cloud in Sara (Amsterdam) via a dedicated 1 Gbit optical line. A schematic representation of the system can be seen in Figure 5-2.

bandwidth performance. http://sourceforge.net/projects/iperf/

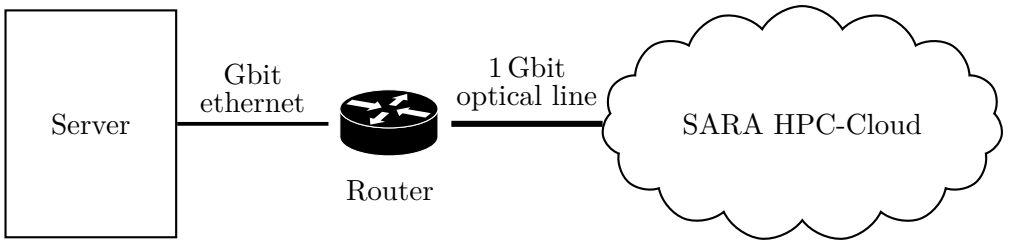


Figure 5-2: System setup for real world experiment using one IaaS cloud.

5-2-2 Deploying the simulation experiment on a real cloud

Experimental setup

As a first approach, we have considered the same users as the case was in the simulation approach in Chapter 4 (Table 4-2). The users and the scheduler were active for a period of 9 hours.

The users were generating datasets once an hour. Using the FIFO link allocation policy, the datasets were sent to the cloud for analytics. For every dataset, a fixed amount of VM(s) have been launched. For every user, the dataset was split in as many parts as the number of VM(s) that were deployed and by using that many parallel connections the data have been transferred to the cloud. The total number of parallel connections per user (i.e. VM(s) deployed) can be seen in Table 5-2. Because of restrictions, it was not possible throughout the experiments to launch hundreds of VM(s), like the normal procedure would have been for analytics of class $O(n^2)$. For this reason we have only run analytics of class $O(n)$.

User	Parallel transfers
#1	4
#2	2
#3	5
#4	5
#5	6
#6	10

Table 5-2: Parallel transfers per user for the experiment using the same user setup as in the simulation approach.

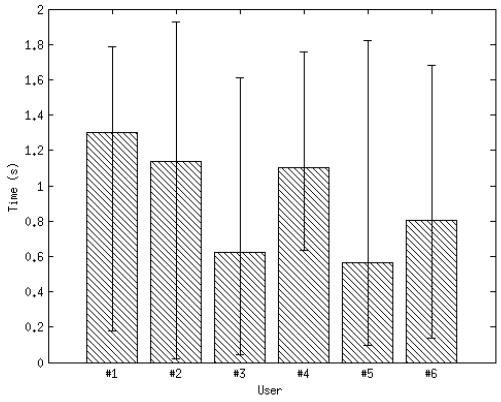
Since the datasets that were used did not push the system to its limits, the waiting times for every user are very low as it is shown in the results provided in Figure5-3. In practice this means that every newly arriving dataset has been finding the waiting queue empty and the link available to transfer the data to the cloud.

For the transmission of the datasets over the network, we have used the Secure Copy Protocol (SCP) protocol. The reason for this is that most of these companies require encryption to avoid leaking sensitive user data. The SCP protocol itself causes high CPU usage which may lead to an enormous overhead for multiple parallel transfers. Our server was a system

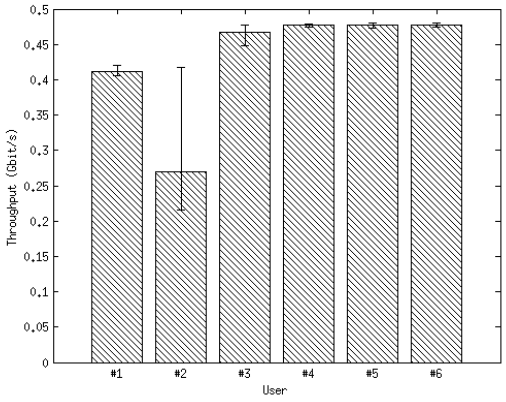
consisting of 4 cores, all of which were fully utilized even when there were 4 parallel connections established.

Results

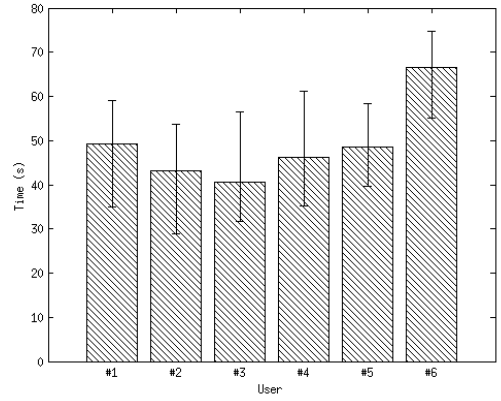
Figure 5-3 shows the results of the experiment. Each bar corresponds to the average value of the particular measurement throughout the experiment. The maximum and minimum values are also represented with the error-bars that go below and above the respective average value.



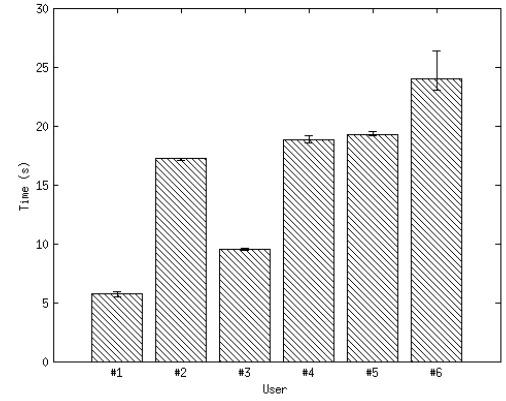
(a) Average waiting time in the queue per user



(b) Average throughput per user



(c) Average VM(s) setup time per user



(d) Average analytics time per user

Figure 5-3: Real world experiment using the same user setup as in the simulation approach and one IaaS cloud.

To get a good grasp of the results we need to understand a few things, some of which have been mentioned throughout this thesis report. Although the link capacity was 1 Gbit, one single file transfer from the our server to the cloud could not utilize the total available throughput. Our experiments have shown that we could not write to a single VM(s) with rates greater

than 200 Mbps. For this reason, we have selected for this experiment a varying number of parallel transfers, to depict this behavior.

The results show that, users establishing more parallel connections have been (in general) able to achieve higher average aggregate throughput. The maximum average attained aggregate throughput throughout the experiments was 0.47 Gbit/s for 5 or more parallel outgoing connections to the same number of VM(s). This means that SCP could manage a link utilization of around 50% if we consider the maximum aggregate throughput that we have measured in subsection 5-1-3.

The waiting time per user is low, which means that every user that had a dataset ready to be sent, kept on finding the queue empty. By using larger datasets or random inter-arrival times for the datasets this would not have been the case.

The average VM(s) setup time per user also varies. From Figure 5-3, it is evident that the formula that was used in Chapter 3 to calculate the time needed to launch a predefined number of VM(s) stands true and it depicts the behavior of a real cloud system. The more VM(s) a user tries to deploy at once, the more time is needed for the whole procedure to take place.

Finally the analytics time is generally small due to the fact that the analytics were of class $O(n)$. The time of the analytics is proportional to the input size in lines of the dataset (after the splitting procedure has taken place).

5-2-3 Efficient dataset transfer using SCP

Private line

As a second experiment, we have considered a dataset of given initial size. Our interest here has focused on the need to transfer this dataset to the cloud and analyze it as fast as possible. What is the optimal way to transfer this dataset over the network and run the analytics on the cloud that minimizes the total time for all the intermediate steps?

We started with the dataset itself and one VM(s). Following, we have increased the number of VM(s) and split the dataset in as many parts as the VM(s) that were deployed (see Table 5-3). The required number of VM(s) has been deployed and terminated for each step of the experiment.

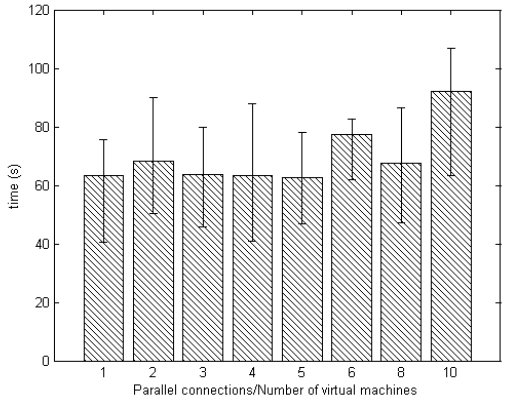
We have considered the same system setup as in the previous experiment. The datasets have been created on a local server and they have been transferred to the cloud using an optical line with a total available nominal throughput of 1 Gbit. For the transfer of the datasets across the network we have utilized the SCP protocol. The initial dataset size that we have considered in this experiment was 6 GB.

Figure 5-4 shows the results of the experiment. We have run the experiment 10 times and the plots that follow are the average results. Error-bars with the minimum and maximum values that have been observed throughout the experiments are also presented.

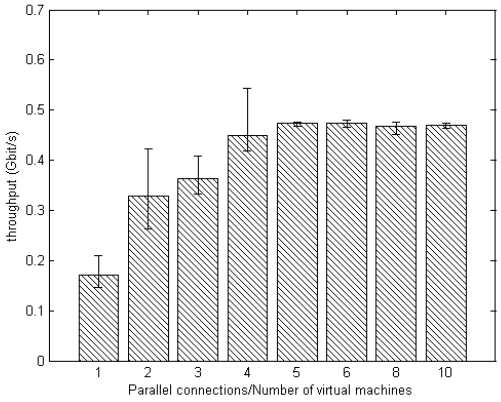
We should note here that all the results depicted in this thesis are actually affected by the hardware specifications of all the system parts under examination. The server on which the scheduler is running as well as the cloud specifications affect the result of the experiments

Case	Deployed VM(s) Parallel Transfers
#1	1
#2	2
#3	3
#4	4
#5	5
#6	6
#7	8
#8	10

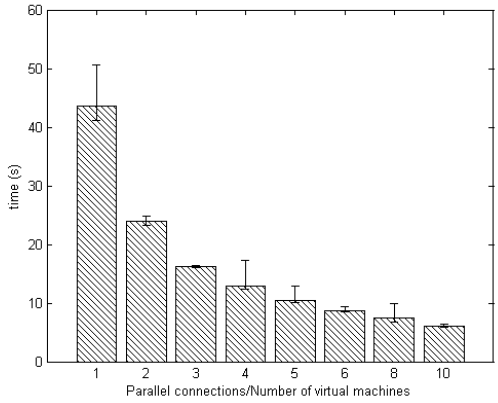
Table 5-3: Deployed VM(s)/Parallel Transfers - Optimal Dataset Transfer Experiment.



(a) Average VM(s) setup time per case



(b) Average throughput per case



(c) Average analytics time per case

Figure 5-4: Efficient dataset transfer using one IaaS cloud over a private link (SCP).

directly. This means that overhead starts to appear when all the resources have been exhausted to all their extent.

Considering the VM(s) setup time, we observe the same behavior as in the previous experiment. The more the VM(s) we are trying to deploy, the more time is needed. For a small number of VM(s) this is not clear, but as we can see in this particular situation, the average time to launch 10 VM(s) is almost 30 seconds larger than the time to launch 5 VM(s) which corresponds to almost 150% of the time needed in the latter case. Although the setup time is almost the same for up to 8 VM(s), we observe a vast increase for the setup time of 10 or more VM(s). The VM(s) setup time is directly affected by the scheduling policy that the cloud uses to deploy the several VM(s) from the various users. It is often the case that the VM(s)

When it comes to the aggregate throughput in each of the test cases, we observe here that it is maximized when using 4 to 6 parallel transfers. Since we have been using SCP, more than 6 parallel transfers increases the demand in computation power that is required on the server part vastly, leading to zero gain in the attained aggregate throughput. This means that with this specific system setup and by utilizing SCP, the total available throughput cannot exceed the previously attained maximal value.

The Internet

The same experiment as in the previous subsection was deployed using a network link that traverses the Internet to reach the cloud. For this experiment, we have considered a dataset size of 2 GB and for the transferring of the data across the network we have utilized the SCP protocol.

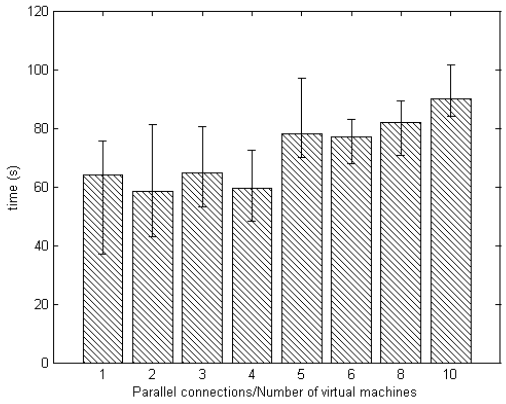
Same as before, we started with the dataset itself and one VM(s). Following, we increased the number of VM(s) and split the dataset in as many parts as the VM(s) that were deployed (see Table 5-3). The required number of VM(s) have been deployed and terminated for each step of the experiment. Finally we run the experiment several times and took the average of all these run-times.

The results can be seen in Figure 5-5. We have run the experiment 10 times and the plots that follow are the average results.

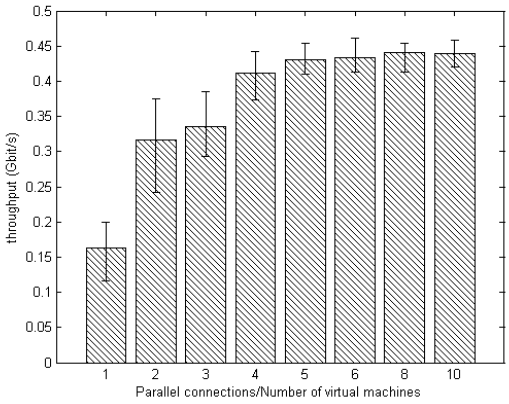
Cosidering the VM(s) setup time, we observe here the same behavior as in the previous experiment. This verifies that the results that concern the setup time of VM(s) are accurate and independent of the network link that we are utilizing.

Since the network speed over the internet in this case was 1 Gbps, we observe, more or less, the same behavior and almost the same results as with the private link case. Since we do not have any statistics on the network usage over the internet for the links that we have been utilizing, it is not safe to reach any conclusion. What is certain is that the network link that we have utilized is a shared connection and the results are the same as when using our own private connection.

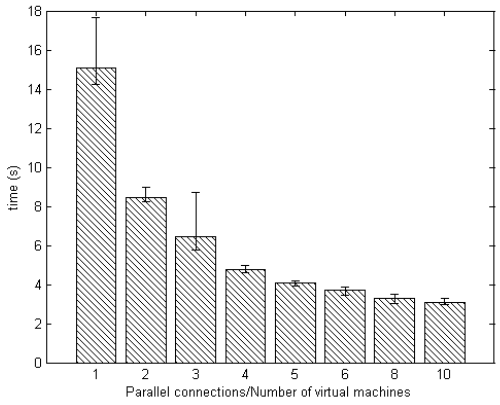
For the average aggregate throughput we have observed the maximum value of 0.43 Gbps. A total of 5 VM(s) were sufficient for reaching this maximum value. The time required to run the analytics is - in every case - 3 times less than the respective cases in the previous subsection. This was an expected result as the dataset that was used was 3 times smaller and the analytics time is independent of the network that has been used to transfer the data to the cloud.



(a) Average VM(s) setup time per case



(b) Average throughput per case



(c) Average analytics time per case

Figure 5-5: Efficient dataset transfer using one IaaS cloud over the Internet (SCP).

5-2-4 Real time analytics using SCP

Depending on the workload, in order to support real time analytics, it is a common requirement that the datasets do not queue up waiting for the datasets that were sent beforehand to finish their analytics procedure. In this subsection we are interested in the size of datasets that can arrive per time interval so that the datasets do not queue up.

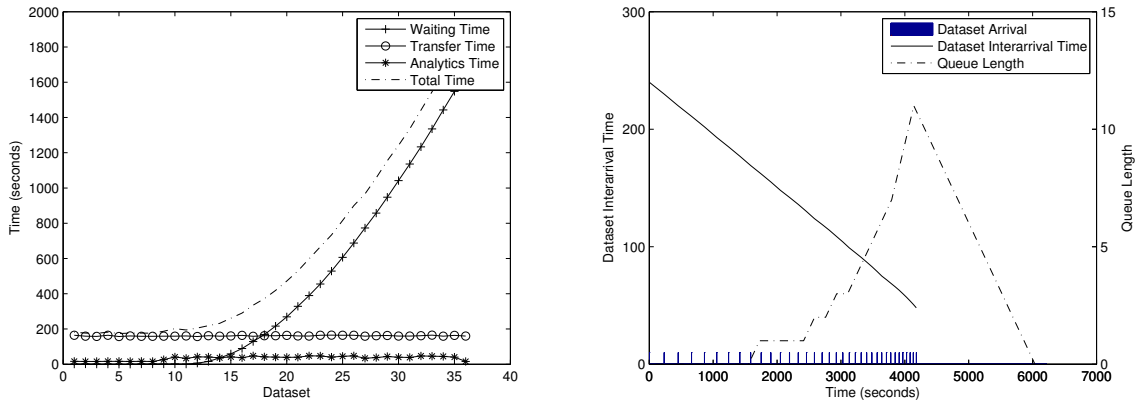
In this experiment, datasets have been generated every a few minutes or seconds. These datasets were being sent over the network to the cloud upon generation. Multiple transfers of different datasets could not have taken place at once, but data transferring could have occurred in parallel with running the analytics on the various deployed VM(s). The inter-arrival time of the datasets has been decreasing as we progressed through the experiment. Finally, the analytics of different datasets could not overlap.

The following experiments were run using the SCP protocol for the transferring of the datasets from the datacenter to the cloud. The experiments were repeated several times in order to

make sure that the results were consistent. We have provided 2 different plots to support our results. The first one contains various performance metrics of interest, like the waiting time, transfer time, analytics time and total time. All the previous metrics are depicted in the first plot over time (dataset arrival timestamp). The second graph contains the arrival timespamps of the datasets of the first plot. Moreover, the interarrival time is shown along with the queue size progression over time. The same plots will be repeated across this chapter for all tests that concern real time analytics.

Private line

We have run this experiment using a 10 GB dataset, and a constant number of 5 VM(s). We run the experiment 5 times and we present here the average of those repetitions. The average aggregate throughput throughout this experiment was 0.47 Gbps.



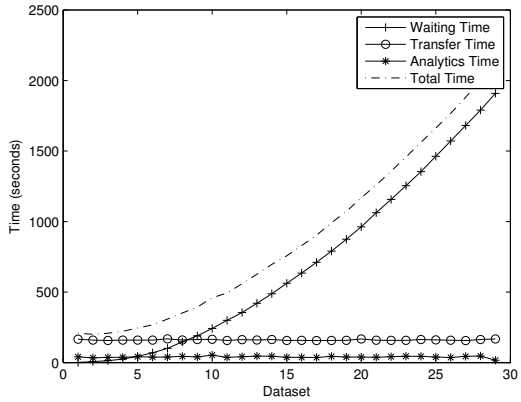
(a) Various system metrics over time

(b) Queue size and inter-arrival time over time

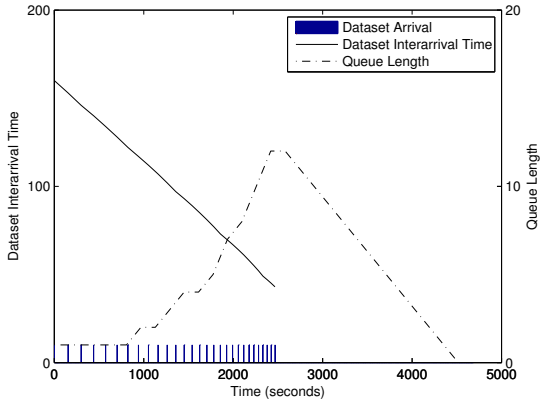
Figure 5-6: Real time analytics experiment using one IaaS cloud over a private line (SCP).

From the results depicted in Figure 5-6, one can find the inter-arrival time for the datasets that can be supported by the system under consideration. This depends, of course, on the dataset size and the number of VM(s) that will be used for the analytics. The result is the same regardless of the size of the dataset we are considering. From the figures above, we have found that an interarrival time of 13 seconds per gigabyte can be supported. The fragmentation of small datasets may lead to some overhead that can change the previous value, but in general the previous statement can be considered as valid.

The above is of great importance when it comes to real time analytics. The results show that roughly, the system can analyze 277 GB of datasets every hour when we are considering analytics of class $O(n)$.



(a) Various system metrics over time



(b) Queue size and inter-arrival time over time

Figure 5-7: Real time analytics experiment using one IaaS cloud over the Internet (SCP).

The Internet

We have run this experiment using a 10 GB dataset, and a constant number of 5 VM(s). We run the experiment 10 times and we present here the average of those repetitions. The average aggregate throughput throughout this experiment was 0.46 Gbps.

From the results depicted in Figure 5-7, one can figure out the inter-arrival time for the datasets that can be supported over the internet for the system that we have considered for our experiments. This depends, like previously, on the dataset size and the number of VM(s) that will be used for the analytics.

We have concluded from the plots above that, on average, the system can support the arrival of 1 GB of dataset every 14 seconds. This means that a total of 257 GB can be analysed in real time when considering the internet for the communication between the datacenter and the cloud.

5-2-5 Efficient dataset transfer over a private line using BBCP and UDT

When transferring sensitive user information over a network link, it is of utmost importance that no information may leak when a successful network eavesdropping attack occurs. All the information that the various companies collect from the user activity is usually protected by the terms and conditions that a user accepts prior to utilizing the service.

SCP is the obvious solution to the previous problem, as it is a well known and widely used protocol for secure data transfer over public networks. As we have seen previously though, SCP could only reach a network utilization of 50%, or at least this was the case for our test system. Undoubtedly, this is a very poor network utilization which is of vital importance when it comes to the requirements of real time analytics.

In order to overcome these issues and reach better network utilization percentage, we have run the previous experiments using BaBar Copy (BBCP)[8] and UDP-based Data Transfer (UDT)[9], two protocols that have been well known for their throughput performance and were specifically designed for the whole purpose of transferring large amount of data over a network at near link rates. For UDT we have made use of version 4.10 of the protocol as it was proposed by its authors.

Due to "closed port" policy in Sara's HPC Cloud for incoming connections from the Internet, the same experiment could not be executed using the Internet connection. Furthermore, the server on which our experimental framework was deployed was behind a Network Address Translation (NAT) firewall with no options to forward the necessary ports on our behalf for security reasons. Because of this, we have only provided in the following paragraphs results using the private interface we had available for which all ports were open and accessible.

BBCP

We ran the efficient dataset transfer experiment of subsection 5-2-3 using BBCP in order to find out how much gain in link utilization we can achieve. The results can be seen in Figure 5-8.

Same as previously, the horizontal axis corresponds to the number of VM(s). BBCP supports multiple parallel connections per transfer with a minimum of 4. In general, BBCP is very difficult to set up due to the many different configuration options that it can take as input. Furthermore, it is very sensitive to small changes with huge throughput variations between different configuration options.

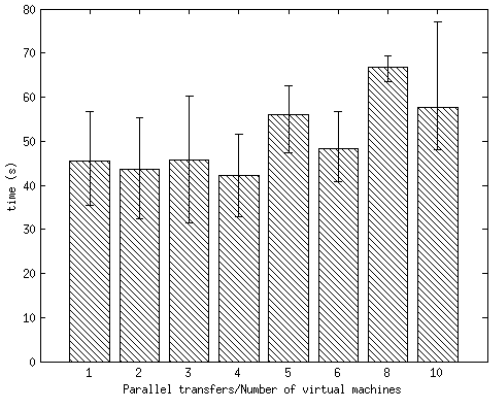
The results show an important increase in the aggregate throughput performance which kept on increasing the more VM(s) we have been using. The largest value for the average aggregate throughput that we have witnessed for 10 VM(s) (10VM(s)$\times$4 parallel connections per transfer = 40 total parallel connections) was 0.76 Gbit/s which corresponds to an 80% link utilization.

For the single machine transfer case there is a huge variation between the largest and the smallest value. Since such a variation could be a result of an error in the experimental procedure, we run the experiments more times to exclude this scenario. Nonetheless, the supplementary experiments led to the same behavior. Even by excluding from the results the highest and lowest values the situation was exactly the same. The reason for this extreme behavior could be a result of the variations we have witnessed in subsection 5-1-3. There, we had considered for this behavior heavy network traffic in the cloud or network virtualization aspects beyond our control. For the average value, a total increase of 30% can be considered an important improvement over SCP although the total link utilization is still as low as 21%.

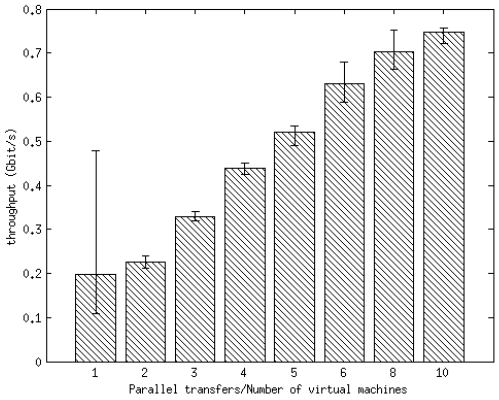
For the remaining tests, which include the average VM(s) setup time (Figure 5-8a) and the average analytics time (Figure 5-8c), the same behavior can be seen as with the test cases using SCP which is exactly as it was expected.

[8] bbcp is a point-to-point network file copy application written by Andy Hanushevsky at Stanford Linear Accelerator Center (SLAC) as a tool for the BaBar collaboration. It is capable of transferring files at approaching line speeds in the WAN [31].

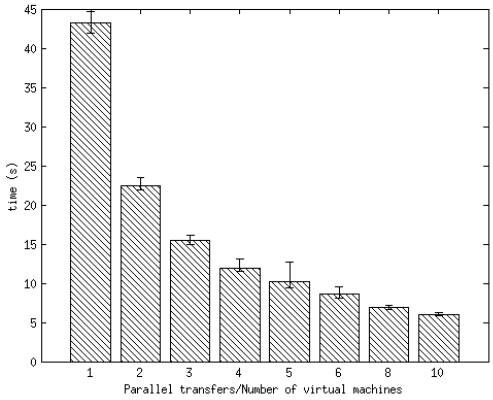
[9] UDT is a high performance data transfer protocol - UDP-based data transfer protocol. It was designed for data intensive applications over high speed wide area networks, to overcome the efficiency and fairness problems of TCP. As its name indicates, UDT is built on top of UDP and it provides both reliable data streaming and messaging services [32] [33] [34].



(a) Average VM(s) setup time per case



(b) Average throughput per case



(c) Average analytics time per case

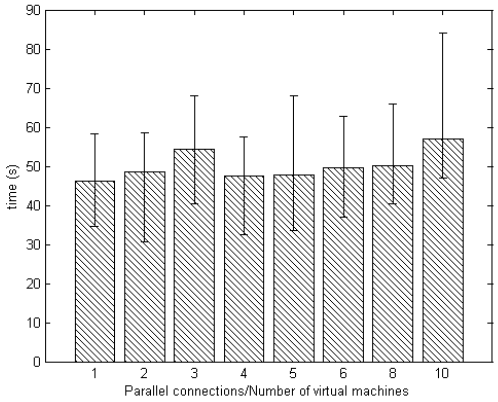
Figure 5-8: Efficient dataset transfer using one IaaS cloud over a private line (BBCP).

UDT4

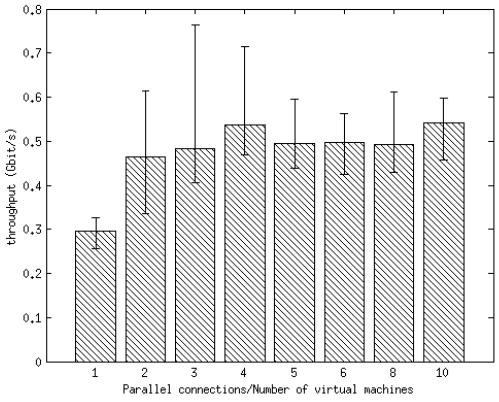
Same as above, we ran the efficient dataset transfer experiment of subsection 5-2-3 using UDT for comparison purposes. The results can be seen in Figure 5-9. The horizontal axes in all of the following figures correspond to the number of VM(s) and subsequently to the number of parallel connections used to transfer data to the cloud since UDT makes use of a single stream of data per transfer. UDT makes use of UDP through a socket interface provided the operating system enabling a UDT socket interface to applications.

With a quick observation of the results, we can see that although we generally get better performance than SCP, the situation is far worse than the BBCP experiment. Not only UDT fails to reach the average aggregate throughput of BBCP but also a huge throughput fluctuation is present.

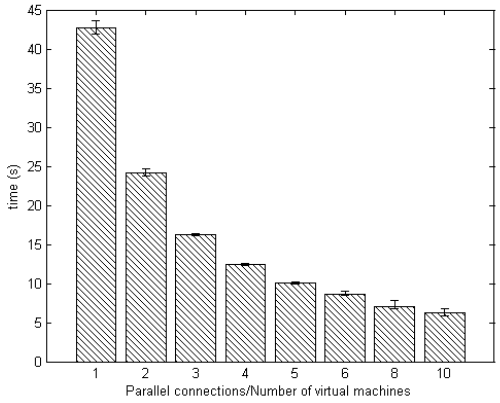
For a relative small number of parallel connections, UDT proves to be faster than the other two protocols. It is extraordinary that UDT manages to reach an average throughput of



(a) Average VM(s) setup time per case



(b) Average throughput per case



(c) Average analytics time per case

Figure 5-9: Efficient dataset transfer using one IaaS cloud over a private line (UDT4).

0.29 Gbit/s when using 1 connection, which can be translated to a link utilization of 32%. This is a total increase of 50% over BBCP and 100% over SCP. Similar comments can be applied for 2 or 3 parallel transfers. A significant downfall in the performance increment is present for more than 4 parallel transfers. This performance degradation could be a result of the centralized concept of our single server which means that UDT could reach a 100% link utilization if more dedicated virtual or native machines were sued to transfer the data to the cloud using the same private link.

For the rest of the metrics, the situation is similar to the previous experiments, both for the analytics and the VM(s) setup time.

5-2-6 Real time analytics using BBCP and UDT

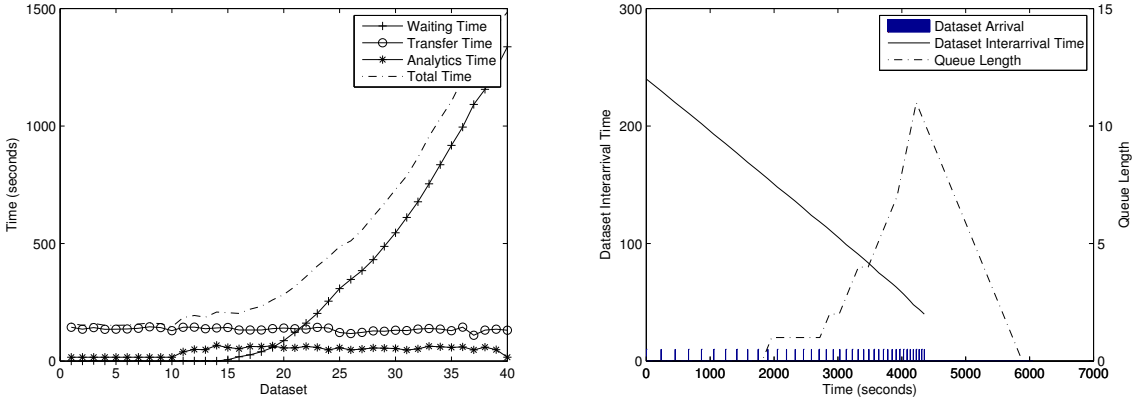
When it comes to real time analytics, using transfer protocols with good link utilization is of major importance. In subsection 5-2-4, we tested SCP which during our experiments

turned out to be a protocol with really poor performance. In this section, we have extended our research of BBCP and UDT to test their performance in our real time analytics test framework that we deployed in subsection 5-2-4.

To get the results provided in the following paragraphs, we have used a 10 GB dataset and a total of VM(s). For each of the test cases, we run the experiments 10 times in order to get consistent results. For each case we have presented 2 different plots like we did in subsection 5-2-4. Finally, all analytics were of class $O(n)$.

BBCP

BBCP was the first of the test cases for this experiment. The results provided in Figure 5-10 show a radical improvement over the usage of SCP.



(a) Various system metrics over time

(b) Queue size and inter-arrival time over time

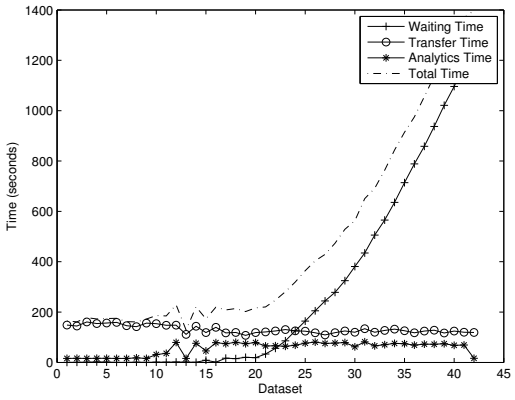
Figure 5-10: Real time analytics experiment using one IaaS cloud over a private line (BBCP).

Our test showed that an arrival rate of 1 GB of data every 11 seconds can be supported, leading to a total of 327 GB of data per hour. Combining this with the results of subsection 5-2-5, the total supported data arrival rate in our system can reach the value of 490 GB per hour when using more than 10 VM(s) at the same time.

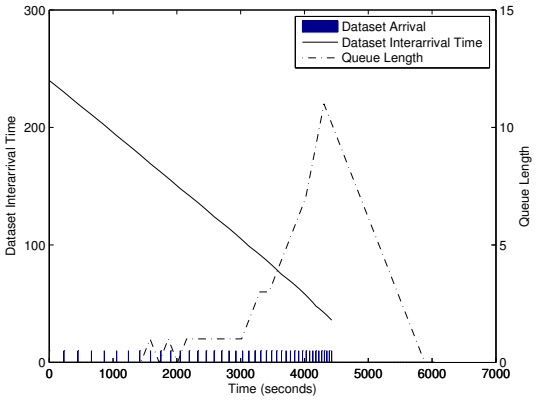
UDT

Our second test case for this subsection was UDT. The results provided in Figure 5-11 show that UDT performed even better than BBCP.

The dataset arrival that was supported before the system became unstable reach the value of 1 GB every 10 seconds. By generalizing the result, we have reached the conclusion that with UDT a total of 360 GB per hour of data arrival can be supported.



(a) Various system metrics over time



(b) Queue size and inter-arrival time over time

Figure 5-11: Real time analytics experiment using one IaaS cloud over a private line (UDT).

5-3 Multiple links

Utilizing multiple links is a great way to increase the system capacity. Multiple links can be used in terms of link aggregation, as a way to increase the total available throughput from one point to the other, or as a way to connect a single point with multiple end-points. We are mostly interested in the second case as it provides us with the ability to connect a centralized computing spot with multiple computation spots which will increase the total system capacity. This does not mean that by doubling the network capacity that it will automatically double the aggregate network throughput. In this section we have studied the presence of multiple links, but we have kept the centralized concept of the datacenter the same.

In the following subsection we are going to see how the system is affected with the addition of one more link.

5-3-1 Real time analytics

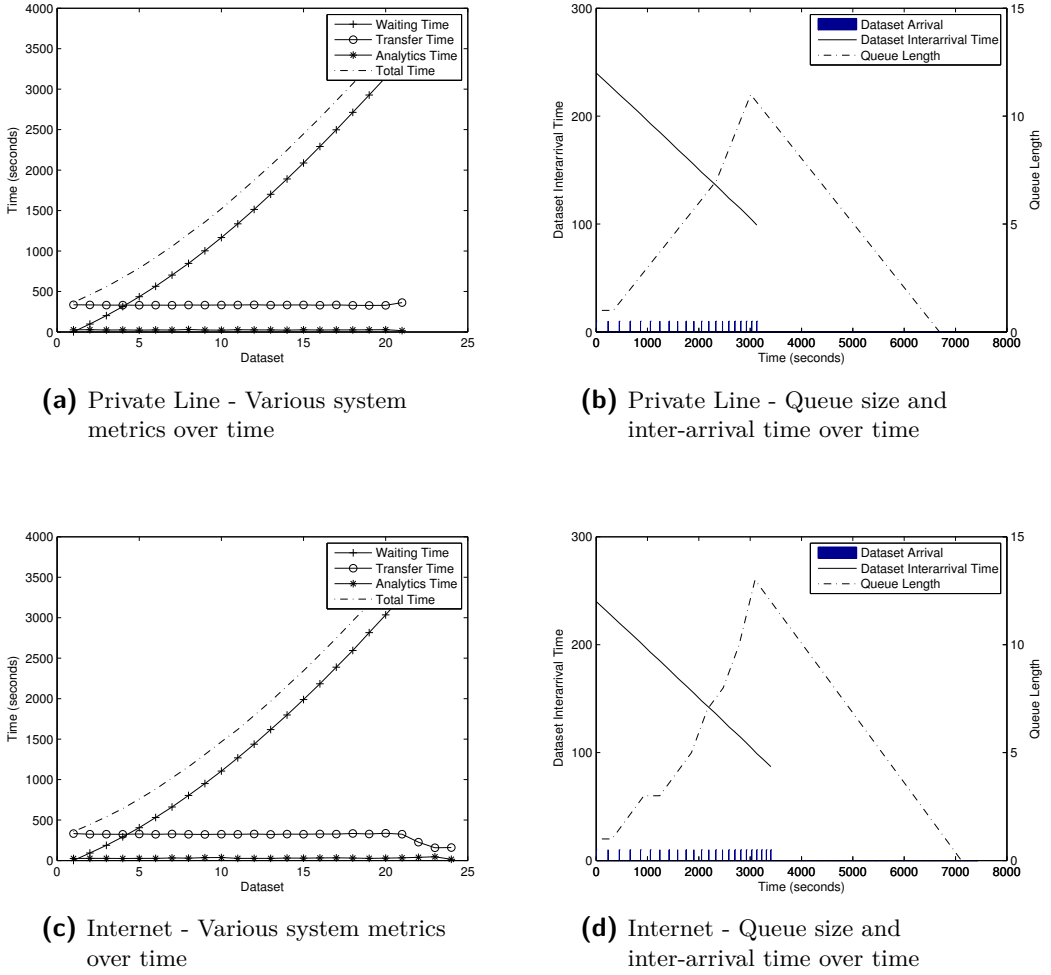
In order to measure the system's performance, we run the real time analytics tests from the previous section in parallel. One has been utilizing the internet as the communication path and the other the private line.

For the private line, a 10 GB dataset was used, same as for the Internet. In both cases, 5 VM(s) where launched beforehand, summing up to a total of 10 VM(s). In any case, the VM(s) that were used to transfer data using the private line did not interfere with those used to transfer data over the Internet.

SCP

During our tests, the total average aggregate system throughput for both links was 0.46 Gbit/s. The two links shared this throughput evenly. Comparing this to the maximum attained throughput which was of equivalent value for the single link case, we have reached the conclusion that when we are using a centralized data transmission spot, SCP cannot perform well even with multiple links. Actually, the presence of the second link is neutralized by the fact that the cap in computational power limits the throughput that total aggregate throughput.

In the following plots the queue size and interarrival time over time can be seen. These plots contain the same information as those in the previous section for the single link case.



(a) Private Line - Various system metrics over time

(b) Private Line - Queue size and inter-arrival time over time

(c) Internet - Various system metrics over time

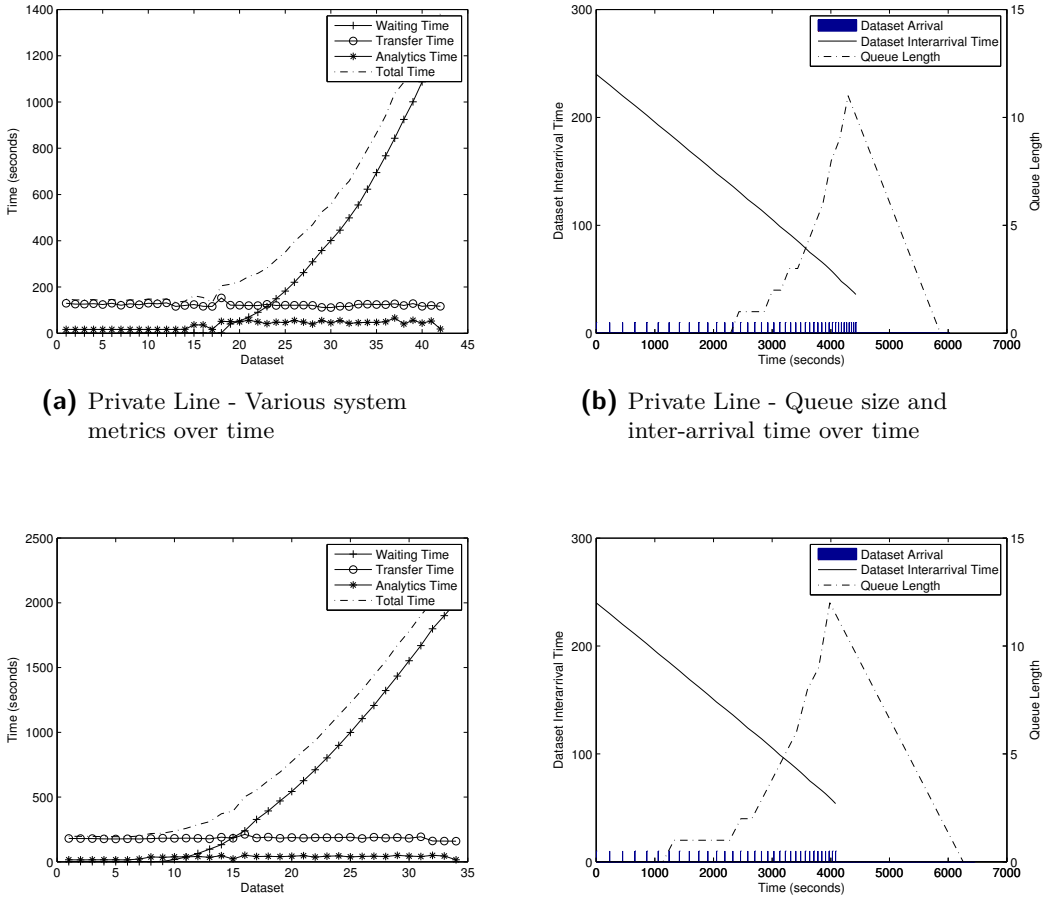
(d) Internet - Queue size and inter-arrival time over time

Figure 5-12: Various systems metrics over time for real time analytics experiment using multiple links (SCP).

By combining the results of the plots shown in Figure 5-12, one can find out the inter-arrival time for the datasets that can be supported by the system for any number of active connections.

From the plots, we have found that an interarrival time of 13 seconds per gigabyte can be supported, same as with the single link case. This means that, roughly, the system can analyze 277 GB of datasets every hour when we are considering analytics of class $O(n)$ for both links.

The previous results lead to the conclusion that for limited processing power in the data transmission spot, SCP is a very poor choice as, the bottleneck that is created due the previous fact, led in our test case to a total link utilization of 25% for both of the links as well as the combination of them.



(a) Private Line - Various system metrics over time

(b) Private Line - Queue size and inter-arrival time over time

(c) Internet - Various system metrics over time

(d) Internet - Queue size and inter-arrival time over time

Figure 5-13: Various systems metrics over time for real time analytics experiment using multiple links (BBCP).

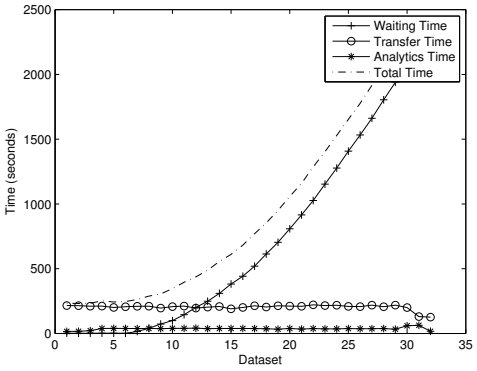
BBCP

We have repeated the following experiment, only in this case we have used BBCP as the transfer protocol for the private link. We have kept SCP over the Internet due to the closed

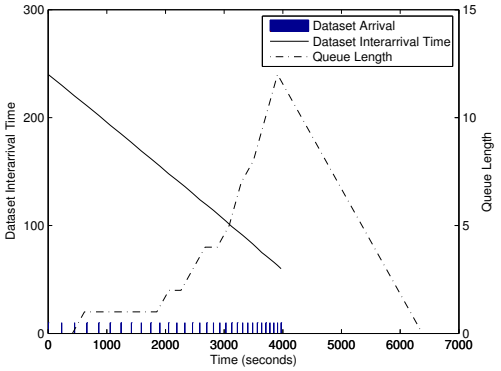
port policy that was explained earlier in the thesis.

By observing the results shown in Figure 5-13, it is noticeable how the system's behavior has greatly enhanced. During this test, the average aggregate throughput when using the private line was 0.61 Gbit/s, while, over the Internet the aggregate throughput's mean value was 0.41 Gbit/s. The previous add up to a total aggregate throughput for the whole system of 1.02 Gbit/s and a total link utilization of 55% for both of the links.

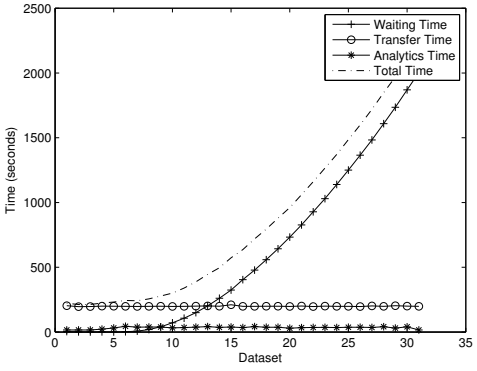
It is noticeable that BBCP performed exactly as it would if the internet link was not active. The further degradation of the performance of SCP leads again to the conclusion that the lack of processing power can highly affect the network performance when using SCP. In this case, the advantage of using multiple links is shown for the first time. The system's performance is almost doubled with a total of 567 GB per hour supported for this test case's setup, which is almost the sum of the individual cases as if they were operating independently.



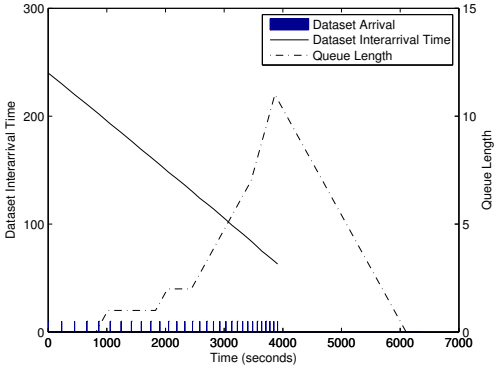
(a) Private Line - Various system metrics over time



(b) Private Line - Queue size and inter-arrival time over time



(c) Internet - Various system metrics over time



(d) Internet - Queue size and inter-arrival time over time

Figure 5-14: Various systems metrics over time for real time analytics experiment using multiple links (UDT).

UDT4

Lastly, we have run the previous experiment, only this time BBCP was replaced by UDT. Although UDT does not use encryption, it seems that it is more computationally demanding than BBCP. This can be supported by the results of this subsection, where we can see from Figure 5-14 that the results are worse than in the previous scenario.

During this test, the two protocols shared the total aggregate throughput that was attained and reached the vallue of 0.75 Gbit/s. The equally distributed throughput can only be a result of a bottleneck in the available processor power. A total of 450 GB per hour have been supported on average for this system setup during our tests.

Setting up a system for real time analytics is a difficult and demanding assignment for an engineer. Because of this, the results shown in the chapter are really important when setting up a real time analytics system. The combination of protocols that are used is important and careful planning is needed. In the case that encryption is required, the situation is even more delicate, as SCP, the most commonly used transfer protocol, has proven to be highly unreliable for real time analytics with poor transfer speeds.

Chapter 6

Conclusion and Future Work

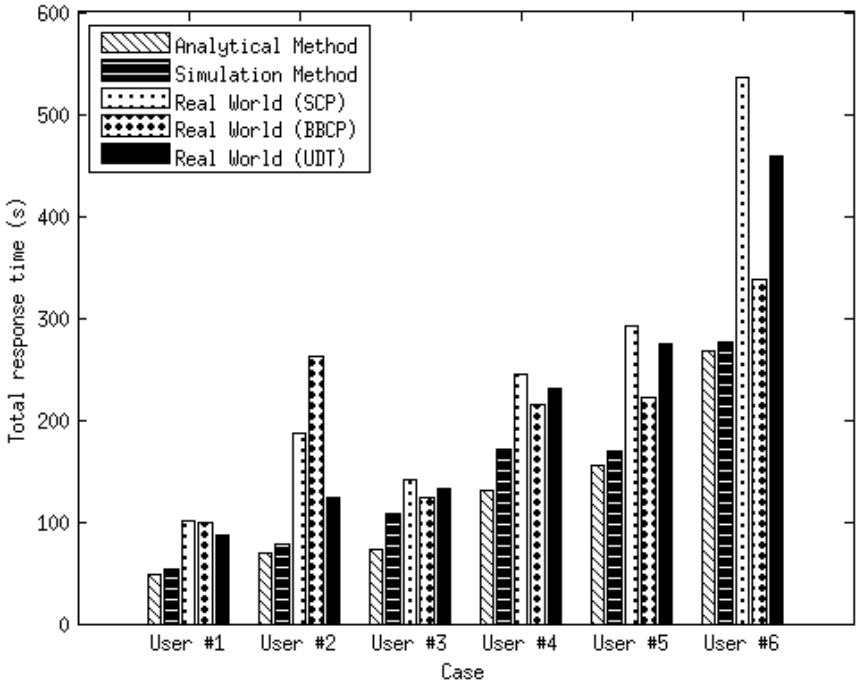


Figure 6-1: Total response time results for the setup of the simulation approach using all the different approaches studied throughput this thesis work.

6-1 Conclusion

This thesis has been a threefold approach to network scheduling for distributed real time analytics of large datasets in the cloud. Threefold as we approached this topic from an analytical (mathematical), a simulation wise and a realistic point of view in terms of real world experimental deployment. Its purpose is to complement existing research topics in cloud computing that hadn't included the network part of the system, when such part exists.

Concluding this work, we have decided to provide as a final proof of concept a comparison among the various parts of this thesis. Using as an entry point the simulation setup that was described in Chapter 4 and further tested in Chapter 5 we have provided in Figure 6-1 the results of this comparison. By observing the plot, the analytical and simulation results are really close to each other. We should note here that we have used the FIFO scheduler to depict the results shown in Figure 6-1. Huge fluctuation may be observed for the real world experimental results, both among the usage of different protocols and among the various approaches. This does not mean that the models presents here are invalid. Regardless of the complication level of the models, the results are surprisingly accurate, with the real world result variation justifiable by the link utilization percentage that was stated throughout the results analysis in Chapter 5. That said, using a link utilization metric for a variety of network transfer protocols, both in the analytic and simulation models, would provide extremely accurate results.

Real time analytics is a really challenging topic, especially with the network in between the data input and the computation spot. Regarding our first research question, we have reached to the conclusion that simple analytics can be executed extremely fast, even by using limited computational power. No matter if the data produced is in terms of a few or several hundred gigabytes, there is no need for the data to be moved to the cloud to get the results back in real time. For more complicated types of analytics, outsourcing the workload to the cloud is the only suitable solution. We have seen that the network is by no means the bottleneck of the total procedure for analytics of time complexity worse than $O(n^2)$.

Throughout this thesis, we have presented models that they have been shown to be accurate for the total response time of large datasets. We have studied several link allocation policies and run several tests in a real cloud using various transfer protocols. The previous were combined in an experimental framework that we used to run analytics in the cloud, in order to see what is the network performance of a real system and under which circumstances real time analytics are feasible. Our results have shown that the network performance is extremely dependent on the system setup. This means that a general purpose cloud for real time analytics is very difficult to be set up and work flawlessly for a variety of users across the world. Nevertheless, the results and models we have presented can be a starting point for solid scheduling mechanisms for real time analytics of large datasets.

6-2 Future work

Our work has focused on a multilevel approach to modelling, simulating and deploying an experimental framework for network scheduling of large from a centralized spot to the cloud. It would be of great interest to utilize different machines inside a core network that would be

used as a transmission point for the datasets, as this is what the real scenario would look like. This way, the insufficient processing power of the system that was used for data transmission would not have been an issue and more accurate results could have been made.

Providing an optimized and even distributed version of our framework is the next step to be made. We have used entry level approaches to schedule the communication of many users, but under no circumstances our approach is the optimal one. Combining the models we have presented along with the results from the performance of the link allocation policies we have tested, as well as the usage of more advanced allocation policies or combining many of them together could result in a really sophisticated scheduling platform.

It was only recently that Amazon has launched *Kinesis*, a service that promises to deal with real-time processing of streaming data at massive scale. A full analysis of how the system operates would be of real interest, as well as the bounds that they can really provide for the response time.

Appendix A

Sara HPC Cloud

A-1 General information

The Sara HPC cloud service was developed by SURFsara as part of the BiG Grid project which ended in December 2012. From 2013 the HPC Cloud services have been taken over and offered by SURFsara, as part of the Dutch national e-infrastructure. With the HPC Cloud, users get access to self-service and dynamically scalable high performance computing facilities[1].

A-2 Technical information

The cloud infrastructure consists of several nodes that are devided into two categories. Depending on the type of virtual machine (VM) a user needs to deploy, the VM is deployed in one of the two categories of nodes.

These categories are:

1. 19 HPC Nodes:
   - CPU Intel 32 cores at 2.13 GHz each (Xeon-E7 'Westmere-EX')
   - 256 GByte RAM memory
   - 1 TByte local disk
   - 4×10 Gigabit Ethernet (GE)
2. 10 'light' Nodes:
   - CPU AMD Opteron™ 8 cores at 2.6 GHz each (Processor 6212)
   - 64 GByte RAM memory
   - 6 TByte local disk
   - 1×10 Gigabit Ethernet (GE)

The network infrastructure consists of 96×10 GE ports, 1-hop with non-blocking interconnect and there is a total of 400 TB available shared storage.

The cloud itself operates using version 3.2.1 of OpenNebula[2] (at the time this thesis project was written), a simple but feature-rich, customizable open-source solution to manage private clouds and datacenter virtualization.

[1] www.surfsara.nl
[2] OpenNebula Project (OpenNebula.org)

Appendix B

Simulation Models

B-1 FIFO

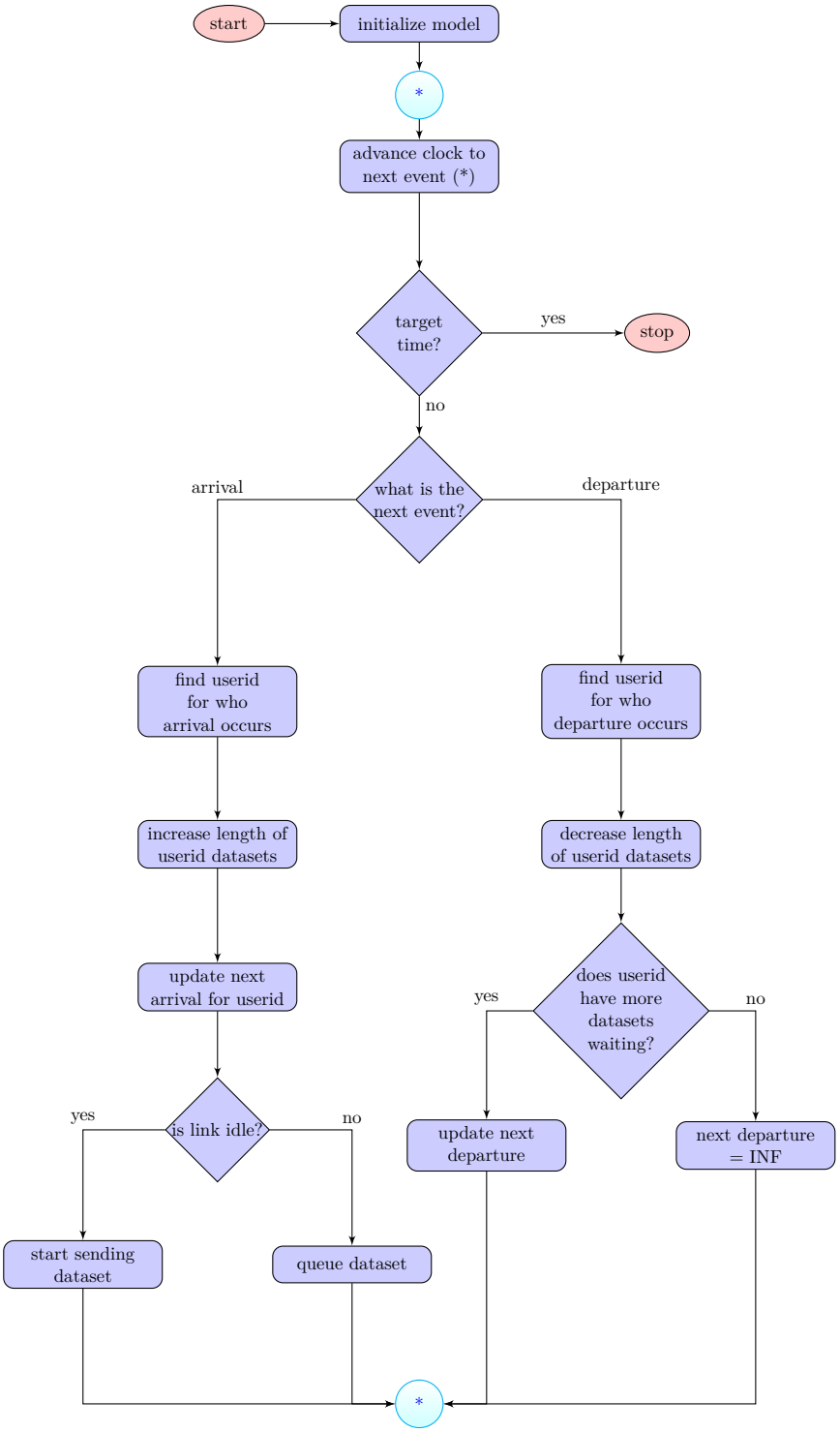


Figure B-1: FIFO queue simulation model.

B-2 ES

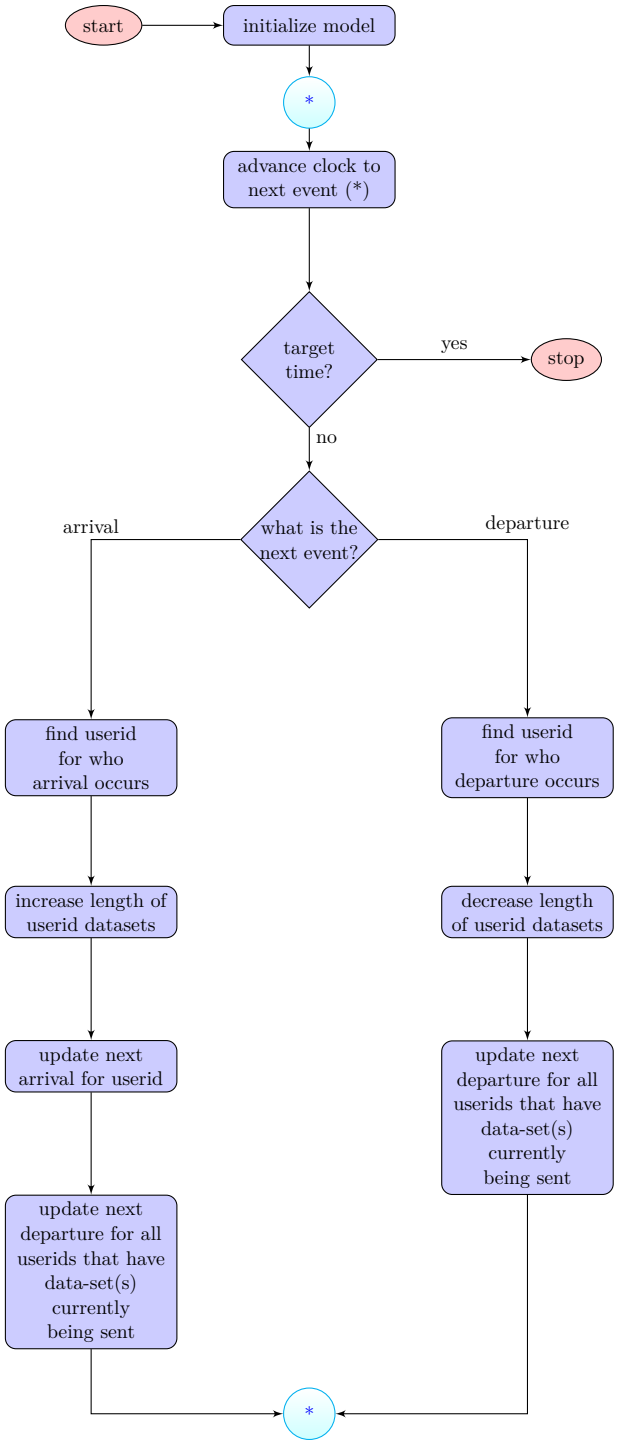


Figure B-2: ES queue simulation model.

B-3 SJF-preemptive

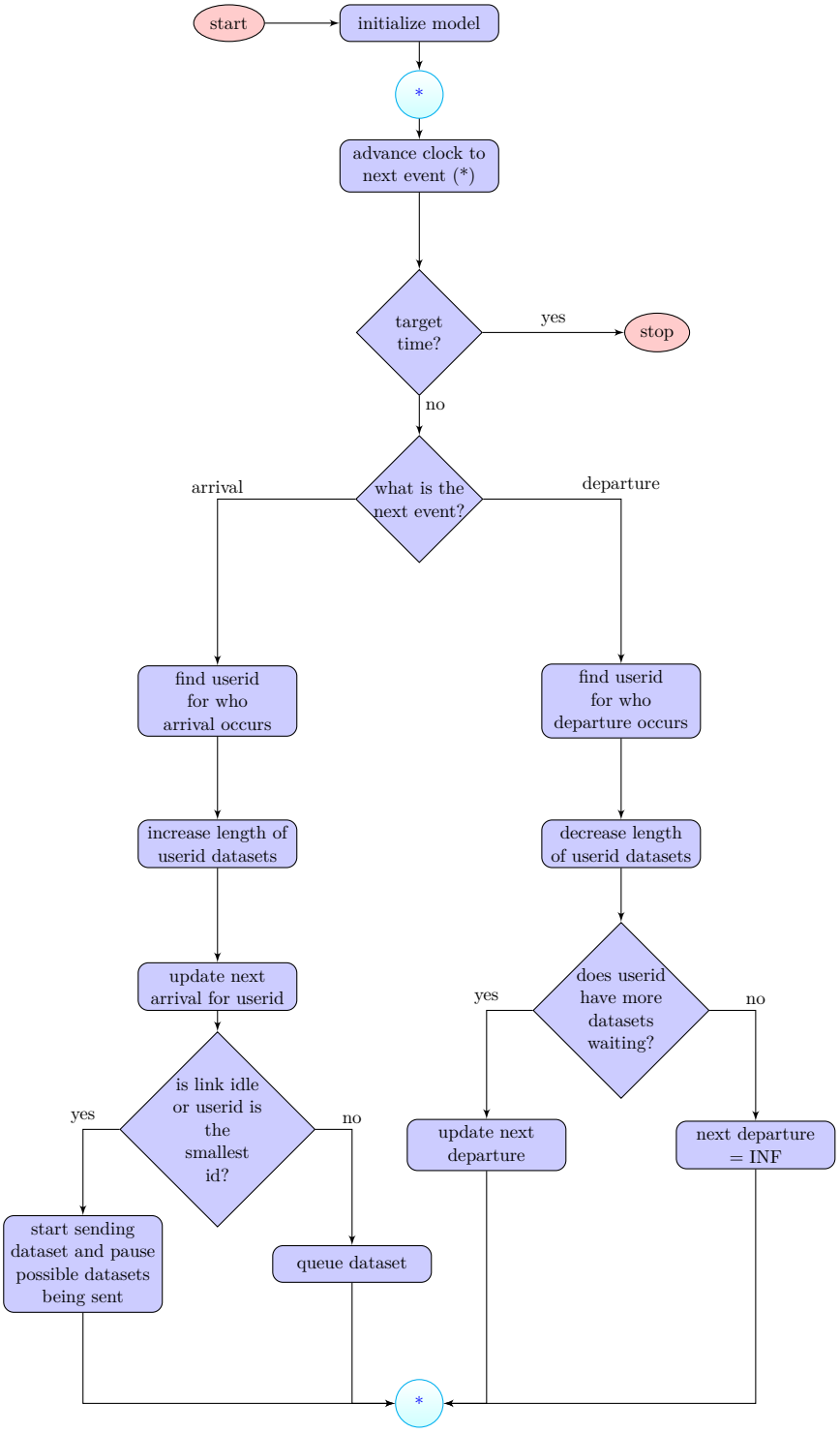


Figure B-3: SJFP queue simulation model.

B-4 SJF

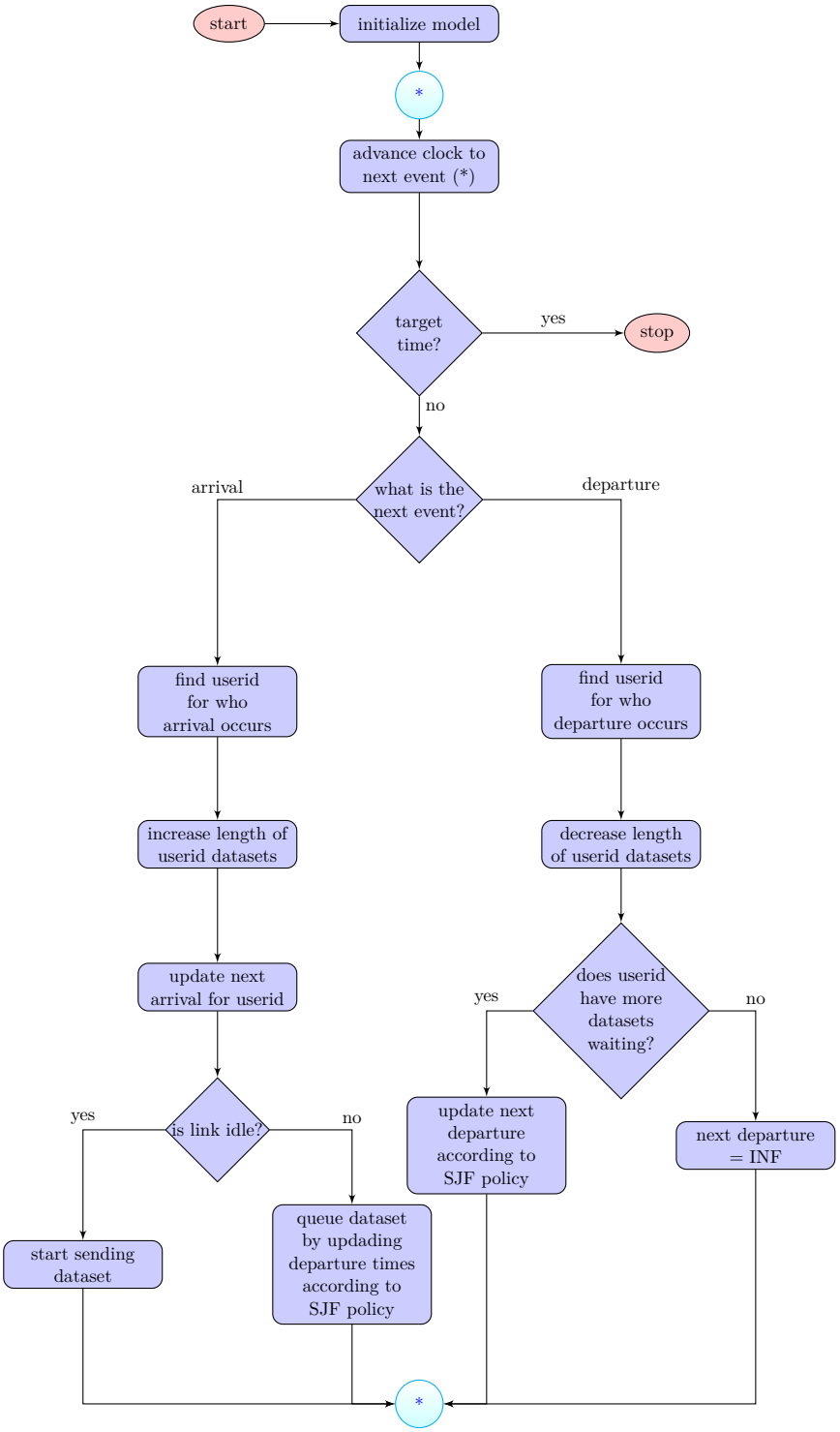


Figure B-4: SJF queue simulation model.

Bibliography

[1] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[2] M. Szell and S. Thurner, "Measuring social dynamics in a massive multiplayer online game," *Social Networks*, vol. 32, no. 4, pp. 313–329, 2010.

[3] R. van de Bovenkamp, S. Shen, A. Iosup, and F. Kuipers, "Understanding and recommending play relationships in online social gaming," in *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*, pp. 1–10, IEEE, 2013.

[4] D. Villegas, A. Antoniou, S. M. Sadjadi, and A. Iosup, "An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pp. 612–619, IEEE, 2012.

[5] A. Iosup, R. Prodan, and D. Epema, "Iaas cloud benchmarking: approaches, challenges, and experience," in *HotTopiCS*, pp. 1–2, 2013.

[6] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pp. 352–358, IEEE, 2002.

[7] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 228–235, IEEE, 2010.

[8] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds," *Future Generation Computer Systems*, 2012.

[9] S. Chaisiri, B.-S. Lee, and D. Niyato, "Optimal virtual machine placement across multiple cloud providers," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pp. 103–110, IEEE, 2009.

[10] F. A. Kuipers, *Quality Of Service Routing In The Internet: Theory, Complexity and Algorithms*. Ios PressInc, 2004.

[11] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 6, pp. 931–945, 2011.

[12] W. Zou, M. Janic, R. Kooij, and F. A. Kuipers, "On the availability of networks," *BroadBand Europe*, pp. 3–6, 2007.

[13] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms*. The MIT press, 2001.

[14] R. van der Pol, M. Bredel, A. Barczyk, B. Overeinder, N. van Adrichem, and F. A. Kuipers, "Experiences with MPTCP in an intercontinental OpenFlow network," June 3-6, 2013.

[15] M. R. Fernandez, "Nodes, sockets, cores and flops, oh, my," November 2011.

[16] G. Paschos, *Introduction to Probability and Stochastic Processes*. 2011.

[17] P. F. A. Van Mieghem, *Performance Analysis of Communications Networks and Systems*. Cambridge University Press, 2006.

[18] I. Adan and J. Resing, *Queueing theory*. Eindhoven University of Technology, 2002.

[19] T. Kleiberg, B. Fu, F. A. Kuipers, P. Van Mieghem, S. Avallone, and B. Quoitin, "Desine: a flow-level qos simulator of networks," in *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, p. 10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[20] R. van de Bovenkamp and F. A. Kuipers, "A toolkit for real-time analysis of dynamic large-scale networks," Proc. of the 20th Annual Symposium on Communications and Vehicular Technology (IEEE SCVT 2013), November 21, 2013.

[21] A. Iosup, A. Lascateu, and N. Tapus, "Cameo: enabling social networks for massively multiplayer online games through continuous analytics and cloud computing," in *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*, pp. 1–6, IEEE, 2010.

[22] M. Perry and A. Nilsson, "Performance modeling of automatic call distributors: Assignable grade of service staffing," *XIV International Switching Symposium*, pp. 294–298, October 1992.

[23] L. Kleinrock, "A delay dependent queue discipline," *Naval Research Logistics Quarterly*, vol. 11, no. 3-4, pp. 329–341, 1964.

[24] L. Kleinrock and R. P. Finkelstein, "Time dependent priority queues," *Operations Research*, vol. 15, no. 1, pp. 104–116, 1967.

[25] S. C. Borst and P. Serri, "Robust algorithms for sharing agents with multiple skills." manuscript, 2000.

[26] N. Gans and Y.-P. Zhou, "A call-routing problem with service-level constraints," *Operations Research*, vol. 51, no. 2, pp. 255–271, 2003.

[27] S. Bhulai and G. Koole, "A queueing model for call blending in call centers," *Automatic Control, IEEE Transactions on*, vol. 48, no. 8, pp. 1434–1438, 2003.

[28] D. Stanford and W. Grassmann, "Bilingual server call centres," *D.R. McDonald and S.R.E. Turner, editors, Call Centres, Traffic and Performance*, vol. 28, pp. 31–48, 2000.

[29] R. Yang, S. Bhulai, and R. van der Mei, "Optimal resource allocation for multiqueue systems with a shared server pool," *Queueing Systems*, vol. 68, no. 2, pp. 133–163, 2011.

[30] B. H. Leitao, "Tuning 10gb network cards on linux," in *Proceedings of the 2009 Linux Symposium*, 2009.

[31] A. Hanushevsky, A. Trunov, and L. Cottrell, "Peer-to-peer computing for secure high performance data copying," in *In Proc. of the 2001 Int. Conf. on Computing in High Energy and Nuclear Physics (CHEP 2001), Beijng*, Citeseer, 2001.

[32] Y. Gu, X. Hong, and R. L. Grossman, "Experiences in design and implementation of a high performance transport protocol," in *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, pp. 22–22, IEEE, 2004.

[33] Y. Gu and R. L. Grossman, "UDT: UDP-based data transfer for high-speed wide area networks," *Computer Networks*, vol. 51, no. 7, pp. 1777–1799, 2007.

[34] Y. Gu and R. Grossman, "UDTv4: Improvements in performance and usability," in *Networks for Grid Applications*, pp. 9–23, Springer, 2009.

Glossary

List of Acronyms

BBCP	BaBar Copy
cpuh	cpu-hours
HPC	High Performance Computing
IaaS	Infrastructure as a Service
MOBA	Multiplayer Online Battle Arena
MMOGs	Massively Multiplayer Online Games
NAS	Network Architecture and Services
QoS	Quality of Service
SCP	Secure Copy Protocol
UDT	UDP-based Data Transfer
VM(s)	Virtual Machine(s)