

High Performance Computing workload traces standardization and characterization

Eneko Pérez Llamazares

Computer Science and Engineering student by The Open
University of Catalonia

BS Final Project

June 2017

Advanced Computer Architecture Department

Open University of Catalonia

High Performance Computing workload traces
standardization and characterization

BS Final Project

Author: Eneko Pérez Llamazares
*Computer Science and Engineering student by The
Open University of Catalonia*

Advisor: Dr. Francesc Guim Bernat
Dr. Computer Science

2017

Title:

High Performance Computing workload traces
standardization and characterization

Author:

Eneko Pérez Llamazares

BS Final Project Committee:

President: PRESIDENT

Speaker: VOCAL

Secretary: SECRETARY

Agree to mark with the qualification:

Barcelona, 25th June, 2017



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

Acknowledgments

Me gustaría agradecer a Francesc Guim la oportunidad que me ha dado para hacer este proyecto y por ayudarme durante la creación del mismo. A Dae-Jin Lee por su paciencia y ayuda. La estadística, sobretodo la mía, no sería la misma sin el. A Víctor, por su incondicional amistad y apoyo. A mi familia, en particular a mis padres y mi sobrina, por sufrir mis cambios de humor. A Beatriz, por haber aguantado probablemente demasiado, y por animarme a hacer la carrera y sacarla adelante. Sin su ejemplo, no hubiese podido conseguirlo.

Abstract

The evolution of the computers allowed us to create supercomputers able to run thousands of jobs at the same time during long periods of time. Behind these jobs there are multiple applications: forecasting, aerodynamics and molecular simulations are just a small example of how supercomputers have changed the world. However, we live in a moment where Big Data is a key aspect in getting better insights of the huge amount of data that is constantly generated. Clusters can provide two types of data: output from jobs and workload traces. This project aims to provide a framework for workload traces analysis and characterization. By performing such analysis, the project tries to establish a standard to define the contents of a workload trace. Besides that, the resulting framework can provide analysis results than can help understanding the data, taking decisions, etc. While analyzing a very specific data set, we concluded that three major group of variables define most of the data set. We also detected a possible anomaly with the data due to abnormal high values in two variables. Finally, a prediction was performed against this data set and we were able to see how future data should be classified.

Contents

Figures Index	8
1. Introduction	9
1.1 Project motivation	9
1.2 Goals and objectives	10
1.3 Materials and Methods	11
1.3.1 Requirements	11
1.3.2 Data	12
1.3.3 Statistical Analysis	12
1.4 Work breakdown	12
1.5 Organization of the BS Final Project	15
2. Overview of workload traces	16
2.1 Workload traces: key concepts of the data	16
2.2 Schema and Data Format	18
3. Case Study: Google Trace	21
3.1 Introduction	21
3.2 Format	22
4. Trace Analysis Implementation	23
4.1 Solution architecture: Installation and setup	23
4.1.1 Installation	24
4.1.2 Additional software	25
4.2 Formatting the data	25
5. Data Analysis	27
5.1 Descriptive Analysis of the Variables	27
5.2 Variables Discarding	33
5.3 Principal Components Analysis (PCA)	34
5.3.1 Introduction	34
5.3.2 Analysis	35
5.4 K-Means	42
5.4.1 Introduction	42
5.4.2 K-Means Test and Train	44
5.5 Interpretation	46
6. Testing	49
6.1 Example	50
7. Conclusion and Future Work	51
7.1 Conclusions	51
7.2 Future Work	52
8. References	53

List of Figures

Figure 1 Gantt chart.....	14
Figure 2 Simplified Gantt chart.....	14
Figure 3 Basic Architecture Schema	23
Figure 4 Sample of data from the Google data set	25
Figure 5 Sample of the data set once normalized	27
Figure 6 Max CPU distribution	30
Figure 7 Max CPU distribution of data	31
Figure 8 Max Memory distribution.....	31
Figure 9 Max Memory distribution of data	32
Figure 10 Variable comparison table	33
Figure 11 Summary of the PCA analysis with 12 components	35
Figure 12 Bar plot of the proportion of the variance	36
Figure 13 Correlation between variables and PC1 and PC2	37
Figure 14 Bar plot of correlation between Principal Components and variables.....	39
Figure 15 Bar plot of correlation between Principal Components and variables.....	40
Figure 16 Variables relationship in a matrix.....	41
Figure 17 Clusters partitioning using K-means	43
Figure 18 Data representation along the clusters.....	44
Figure 19 Training test data.....	45
Figure 20 Mean CPU usage across the different variables	47
Figure 21 Data representation along the clusters with predictive ellipses and components	48
Figure 22 Data representing the Component 1 and 2 values with respect to Clusters	49

Chapter 1

1. Introduction

The beginning of the digital age is considered to be about 2002 and, at this point, storage started to become more digital than analog. Storing information digitally provided a faster and easier way to use it. However, since the digital age began, digital data sets have been increasing considerably to the point of being too large and complex to be processed with traditional methods.

The concept of Big Data, a term used since the 1990s, pretends to provide techniques and technologies to reveal insights from complex and of a massive scale data sets.

In line with the evolution of the digital data, supercomputers, computers with a high level of computing performance, have evolved and increased their capabilities as well. Many applications, such as weather forecasting, probabilistic analysis, nuclear tests simulations and molecular dynamics simulation, use extensively this kind of supercomputers, coining the term High Performance Computing.

In order to run an application in these computers, a job has to be created and submitted to a job scheduler that will decide where to run the application, depending on a set of constraints.

Apart from the corresponding output of each job sent to a supercomputer, clusters administrators have access to workload traces. A trace is usually a time stamped sequence of events captured on a computer with tools designed for that purpose.

Traces are significantly useful, for example in order to make an analysis of the different jobs running on a cluster. This kind of analysis can help make decisions regarding new acquisitions, software configuration, job scheduler setup, scalability, etc.

1.1 Project motivation

Many different cluster applications lead to different workload traces. These, and the lack of a standard workload trace format, force us to analyze each case separately. This entails different drawbacks, such as difficulty to

compare results and additional previous work of analysis. In addition, workload traces are not usually publicly available.

However, one of the largest companies in the world, as well as one of the most important technological companies, Google, made available traces from their supercomputers. Due to the complexity of its infrastructure and the inherent problems related to it, such as dealing with millions of different jobs on thousands of different machines, analyzing the data of their clusters may provide a valuable insight for making decisions in areas like acquisition, cooling, software and systems configuration and more.

Google has been publishing what they call ‘Cluster workload traces’, making available two versions of this data set. The first one, ‘TraceVersion 1’ is an older, shorter trace describing a 7-hour period from one cluster. It is however deprecated, and they encourage using the newer version, ‘ClusterData2011_2’. This data set provides data from a cluster of more than twelve thousand machines for about a month period in 2011.

When we analyze information of a cluster, we instinctively think in a few components of the system: processors, memory and storage. These are the three pillars on which, we think, modern computers are seated. However, partly thanks to the Big Data, we are capable now to extract valuable information of a trace data set, and get an insight view beyond CPU, memory and disks. For all of the above, we believe that a company like Google is the perfect target for such kind of analysis.

Our motivation in this BS Final Project will be analyzing the traces made available by Google, and offering an insight view of the real important data inside. During this process, and due to a prior analysis, we will try to propose a standardized workload trace format as well.

1.2 Goals and objectives

The main goal of this BS Final Project is to create a framework that can analyze an input trace, as well as characterize the main metrics. The following objectives have to be accomplished:

- Identify common and relevant fields in the trace by using statistical analysis of the data.
- Propose a new format thanks to the prior analysis.
- Program a code that can analyze the information and extract key components.

- Program a code that can create graphical characterization of the above.
- Provide a framework to facilitate trace based simulation studies leveraging a set of analysis that can be used prior to simulation to quickly understand and characterize traces.

1.3 Materials and Methods

We describe the data that we use, the statistical analysis methods used and the requirements of the process.

1.3.1 Requirements

To achieve the goals and objectives described above, we will need a standard computer or virtual machine running Linux operating system. A description of additional software is shown below:

- Ubuntu Linux operating system or similar.
- R – The R Project for Statistical Computing.

In order to provide a reference, here are detailed the specifications of the computer used for this Final Project. However, the same results should be expected while using modern standard equipment, such as Intel Core I3 or Core I5 processors and a maximum of 8GB of RAM memory. Keep in mind though that if we wanted to analyze a larger data set (or the complete data set published by Google), better specifications would be needed:

- Computer: Dell XPS 13” Developer Edition
- Processor: Intel I7-7560U.
- Memory (MB): 16000.
- Disk (GB): 512 NVMe.

1.3.2 Data

The data for this project comes from a publicly available repository of Google. This data is composed by a series of tables, each of one representing a different data set with different variables. Despite the number of available tables, we decided to use one of them due to the number and type of variables that it has.

The other tables have information about machines, job and tasks and task constraints. However, as we mentioned earlier, the task resource usage table is the most relevant of the data set, and we will focus our project on this one.

1.3.3 Statistical Analysis

Due to the nature of the data, originally composed by 20 variables, we decided to use a statistical method to reduce the dimension of the data. This method, called Principal Components Analysis, is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possible correlated variables into a set of values of linearly uncorrelated variables called principal components. After applying the method, the dimension of the data set is reduced significantly and a few principal components explain the majority of the data.

Besides that, we use a method for clustering and grouping data, known as k-means, as well as a training method for predicting future behavior. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean.

1.4 Work breakdown

The work breakdown is shown below:

- Phase 0: Work breakdown and research (22/02/2017–19/03/2017, 26 days)
 - Creating a work breakdown plan (BS Final Project – PEC 1)
 - Information collection
 - Information analysis

- Phase 1: Installation, Documentation and Analysis (19/03/2017–12/04/2017, 25 days)
 - Installing and setting up the environment
 - Analysis and solutions proposal
 - Documentation

- Phase 2: Design and implementation of the solution (12/04/2017–06/05/2017, 25 days)
 - Designing and implementing the solution

- Phase 3: Analysis and tests (06/05/2017–17/05/2017, 12 days)
 - Analyzing the solution
 - Final testing

- Phase 4: Final memory (17/05/2017–25/06/2017, 35 days)
 - Writing Final Degree Project dissertation

Task name		Start date	End date	Duration day
[-] Total Estimate		22/02/2017 08:00	25/06/2017 09:00	123.04
1	[-] HPC Workload Traces Standardization and Character	22/02/2017 08:00	25/06/2017 09:00	123.04
1.1	[-] Phase 0: Work Breakdown and Research	22/02/2017 08:00	19/03/2017 09:00	25.04
1.1.1	Creating a work breakdown plan	22/02/2017 08:00	26/02/2017 00:00	3.67
1.1.2	Information collection	26/02/2017 09:00	12/03/2017 09:00	14.00
1.1.3	Information Analysis	12/03/2017 09:00	19/03/2017 09:00	7.00
	+ Add a task + Add a milestone			
1.2	[-] Phase 1: Installation, Documentation and Analysis	19/03/2017 00:00	12/04/2017 09:00	24.38
1.2.1	Installing and setting up the environment	19/03/2017 00:00	21/03/2017 04:00	2.17
1.2.2	Analysis and solutions proposal	21/03/2017 09:00	05/04/2017 10:00	15.04
1.2.3	Documentation	05/04/2017 09:00	12/04/2017 09:00	7.00
	+ Add a task + Add a milestone			
1.3	[-] Phase 2: Design and Implementation	12/04/2017 09:00	06/05/2017 08:00	23.96
1.3.1	Designing and implementing the solution	12/04/2017 09:00	06/05/2017 08:00	23.96
	+ Add a task + Add a milestone			
1.4	[-] Phase 3: Analysis and tests	06/05/2017 08:00	23/05/2017 09:00	17.04
1.4.1	Analyzing the solution	06/05/2017 08:00	20/05/2017 08:00	14.00
1.4.2	Final Testing	20/05/2017 09:00	23/05/2017 09:00	3.00
	+ Add a task + Add a milestone			
1.5	[-] Phase 4: Final Memory	23/05/2017 08:00	25/06/2017 09:00	33.04
1.5.1	Writing Final Degree Project Dissertation	23/05/2017 08:00	25/06/2017 09:00	33.04

Figure 1 Gantt chart

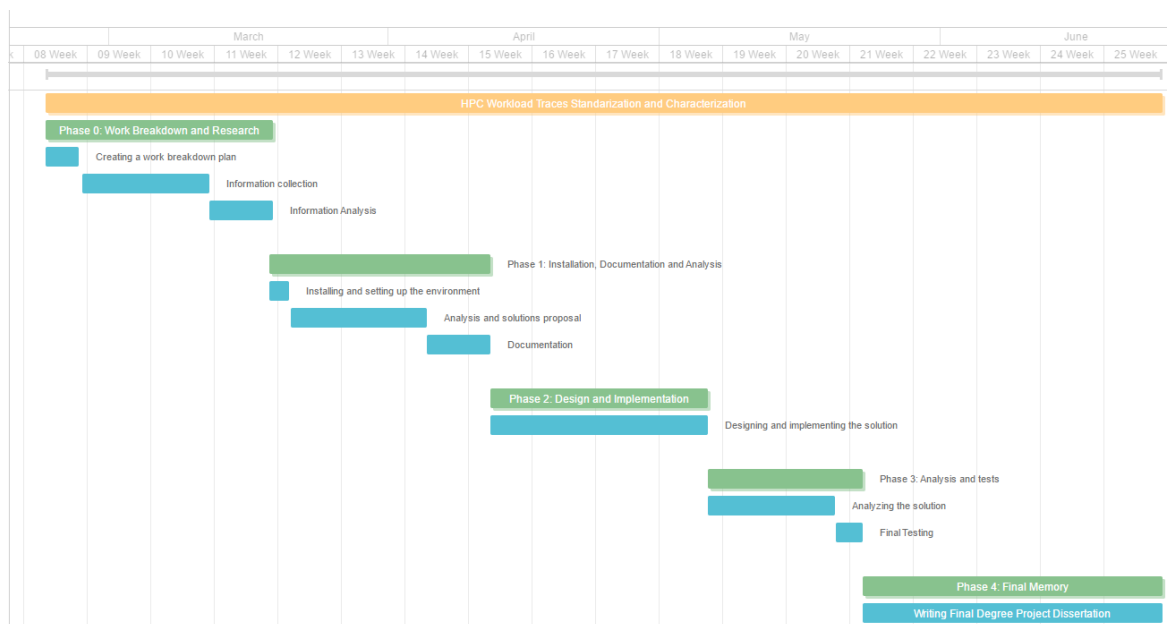


Figure 2 Simplified Gantt chart

1.5 Organization of the BS Final Project

The rest of this BS Final Project is organized as follows:

- **Chapter 2** presents an overview of the workload traces data. We describe workload traces key concepts as well as characteristics. Furthermore, we aim to provide a general overview of the different traces schema and data format.
- **Chapter 3** describes our case study. We discuss about the Google traces. The main goal is to dissect the traces format and try to understand their metrics.
- **Chapter 4** presents our trace analysis architecture implementation. In the first part, we show the solution architecture. In the second part, we explain the procedure of install the different applications and libraries that form the framework that we have developed.
- **Chapter 5** describes the analysis that we have made to the traces, more specifically the Principal Component Analysis (PCA) that it has been performed to the sample of data. Based on the PCA results, we do a profile for a potential standard workload trace format. Besides that, we perform a K-means analysis in order to group or cluster the different tasks in our data set and perform a prediction.
- **Chapter 6** describes the tests performed to the application. The main goal is to check the code working with different traces with the same format automatically.
- **Chapter 7** presents our final conclusions and possible future studies.

Chapter 2

2. Overview of workload traces

In this chapter we present different workload traces found on the Internet. Besides that, we analyze their key concepts and schema.

2.1 Workload traces: key concepts of the data

During the research phase of this Final Project, we have found some workload traces with different formats, metrics and purposes. These traces belong to Google, Carnegie Mellon University, University of California Santa Barbara (UC Santa Barbara), Masaryk University and Facebook. We describe here the main characteristics of every of them.

Google

'Google Clusterdata' is a large-scale production workload trace made available by Google. These are production workloads running on Google clusters. The workload consists of a set of tasks, where each task belongs to a single job and a single job can have one or more tasks

The trace is made up of several data sets. Each data set contains different tables and every table has its own format and represents one aspect of the trace, as we will see in detail shortly. Data is highly anonymized.

Carnegie Mellon University

This workload trace contains Hadoop logs from OpenCloud cluster. OpenCloud is a research cluster at Carnegie Mellon University managed by the CMU Parallel Data Lab. Like in the previous example, data is highly anonymized. The data set is significantly smaller than Google's.

The trace is made up of several tables, each table containing different type of data. There are mainly two types of logs of Hadoop: job configuration and job history files.

UC Santa Barbara

This workload is composed of several files, each of one contains a trace of start and stop events recorded by Eucalyptus Cluster Controller. Eucalyptus is a computer software for building Amazon Web Services-compatible private and hybrid cloud computing environments and it is marketed by the company Eucalyptus Systems. As before, data is anonymized.

Masaryk University

This data set represents the job description and machine description provided by TORQUE job scheduler during a period of 4 months in 2015. The job log is provided in a per-user format but suitable for dynamic workload simulations, fulfilling the specifications of the Standard Workflow Format. Besides that, the log format is compatible with the Alea jobs scheduling simulator.

Facebook

These workloads represent two periods of the same cluster at Facebook. FB-2009 has historical Hadoop traces of a 600-machine cluster, while FB-2010 has historical Hadoop traces of the same upgraded cluster, composed by 3000 machines at that moment. Hadoop is an open source software framework used for distributed storage and processing of a data set of big data using the MapReduce programming model.

Summary

Besides these public workload traces, there is another trace, or more specifically a proposed standard, that is worth studying and mentioning here. The Standard Workflow Format, SWF, was defined to provide a single format of workload that could then be applied in simulation or analysis.

The Standard Workload Format was proposed by David Talby and improved by Dror Feitelson, James Patton Jones, and others.

Along with the information about the SWF, the School of Computer Science and Engineering of the Hebrew University of Jerusalem offers a Parallel Workloads Archive. This archive provides with several traces from different sources already adapted to SWF.

Unfortunately, this archive is not frequently updated.

2.2 Schema and Data Format

Google

Several tables compose Google cluster data set. Once analyzed, we decided to use the table named **Task Resource Usage**, since this is the one giving us more information. Google provides 500 compressed files, each of one containing millions of entries following the schema of this table. This table is composed by 20 variables.

As we will see later, some of these variables are not appropriate for the analysis that we will perform.

Carnegie Mellon University

This trace is made of several tables as well. More specifically, it is composed by the following tables:

- Job Configuration: contains fields that help to uniquely identify a MapReduce job.
- Job History: basically contains the status of a job and the number of map and reduce tasks in the job.
- Task History: the type of task, status and start and finish time.
- Task Attempt History: host where the task attempts to run, job id and time stamp.
- Counter History: Historical data of counters for different type of events.
- Split History: very briefly documented.

The most relevant table of the data set is Job History, since it has information about the timestamp of submitted jobs, status, total number of map and reduce tasks, etc.

UC Santa Barbara

This data set is simpler than those that we have described so far. Each file contains a trace of the VM start and stop events recorded by a Eucalyptus Cluster Controller. Besides the timestamp, there are several useful columns. The start record format is:

START timestamp instance-id node-name core-count

The stop record format is:

STOP timestamp instance-id

Each instance-id should have both a START and STOP record in each file.

Masaryk University

The data set provided by this University follows the Standard Workflow Format (URL). The SWF has 18 data fields. These are the data fields in order of appearance in a log:

Job Number, Submit Time, Wait Time, Run Time, Number of Allocated Processors, Average CPU Time Used, Used Memory, Requested Number of Processors, Requested Time, Requested Memory, Status, User ID, Group ID, Executable Number, Queue Number, Partition Number, Preceding Job Number and Think Time from Preceding Job.

The log format is compatible with the Alea Jobs Scheduling Simulator.

Facebook

These workloads are previously synthesized. They are one day in duration and contain 24 historical trace samples, each of one hour long. The format is:

- New_unique_job_id
- Submit_time_seconds
- Inter_job_submit_gap_seconds
- Map_input_bytes

- Shuffle_bytes
- Reduce_output_bytes
- Anonymized_input_path

Summary

After evaluating the different traces shown above, we believe that those coming from Google can offer more detailed information about the jobs their users run. Google provides much more information and it has many more variables as well. Besides that, everything is very well documented.

Traces Comparison Table

Trace	Features
Google	Dedicated table to task usage resources with 20 variables.
Carnegie Mellon University	Very specific data about map and reduce tasks.
UC Santa Barbara	Time-series data with processor information but lacking of memory or disk related data.
Masaryk University	Interesting data with many variables and following the standard workload format. Still, it presents fewer variables than the Google's data set.
Facebook	Lack of information about processor, memory or disk/storage usage.

Chapter 3

3. Case Study: Google Trace

In this chapter we present our case study. We discuss about the Google traces mentioned before. The main goal is to dissect the traces format and try to understand their metrics.

3.1 Introduction

Usually engineers and researchers in industry have access to large-scale data. At the same time, researchers in academic suffer from a lack of publicly available data from the industry.

Google is one of the companies making publicly available this kind of valuable information. Their latest data, on which we based this project, represents 29 days of cell information from May 2011 on a cluster of 12.500 machines.

As we have seen in the previous chapter, there are several possibilities that offered important metrics like CPU and memory usage, storage, timestamps, etc. However, we found Google's data set the most complete and well documented. Besides that, Google is one of the biggest and one of the most complicated too, in terms of complexity of the infrastructure.

Taking everything into account, we considered that having a better insight of this data would worth it.

However, this data is not free of errors that we should consider while performing the analysis and concluding. For instance, according to Google's documentation, some cycles per instruction (CPI) and memory accesses per instruction (MAI) measurements are clearly inaccurate. This is probably caused by the data-capture system used. This behavior should be explained later in this work.

3.2 Format

This a detailed description of the fields that are part of the Task Resource Usage table:

- Start time of the measurement period
- End time of the measurement period
- Job ID
- Task Index
- Machine ID
- Mean CPU Usage Rate
- Canonical Memory Usage
- Assigned Memory Usage
- Unmapped page cache memory usage
- Total page cache memory usage
- Maximum memory usage
- Mean disk I/O time
- Mean local disk space used
- Maximum CPU Usage
- Maximum disk I/O time
- Cycles per Instruction (CPI)
- Memory Accesses per Instruction (MAI)
- Sample Portion
- Aggregation Type
- Sampled CPU usage

Chapter 4

4. Trace Analysis Implementation

In this chapter we present our data analysis software implementation. In the first part, we show the requirements of the infrastructure and the installation and configuration procedure. In the second part, we perform data formatting; the procedure includes removing unwanted variables and renaming the others. Once we finish this chapter, the data will be ready to be analyzed.

4.1 Solution architecture: Installation and setup

Our solution is very simple in terms of requirements both hardware and software. It is basically composed by three big pieces of software: the data itself, R and the Operating System on which everything runs.

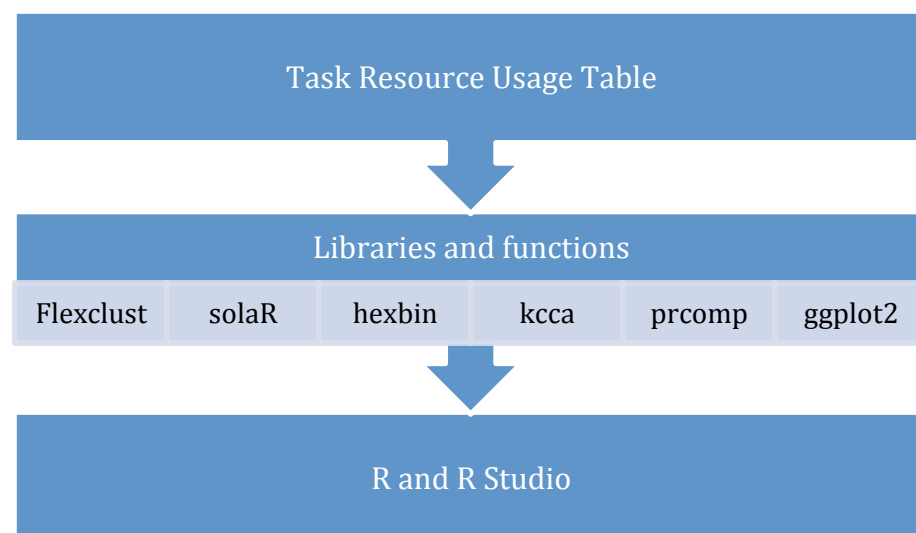


Figure 3 Basic Architecture Schema

R is a free software environment for statistical computing and graphics.

Besides R, the following libraries have been used in this project:

- solaR
- hexbin
- flexclust
- ggplot2

SolaR library is used along with hexbin to plot a scattered matrix that represents the relationships of the variables. Flexclust is provides algorithms to perform clustering analysis. Ggplot2 is used to plot some different graphs.

4.1.1 Installation

We decided to use an standard operating system such as Ubuntu 16.04 LTS.

On top of the operating system, we installed R and R studio. R requires about 1GB of RAM and a non-root with **sudo** privileges in the system.

R

The first step is to add the R repository to our Linux system. We need to create an entry in */etc/apt/sources.list* with this information:

```
deb https://cloud.r-project.org/bin/linux/ubuntu xernial/
```

After this step, we are ready to continue by updating the system so it refreshes the packages available at the recently added repository. From a terminal, write the following commands:

```
sudo apt-get update
sudo apt-get install r-base
```

After running these commands, R will be installed in our system.

Libraries

In order to install the libraries that we mentioned before, these steps should be followed.

From a terminal, launch R by typing 'R' and pressing 'enter'. Once we are inside the R interpreter, we proceed to install the packages:


```
install.packages(c("solar", "hexbin", "flexclust", "ggplot2"))
```

The installation process would probably ask the user to select a mirror from which download required packages and their dependencies. If that is the case, mirror 1, "cloud mirror", it is a good choice.

The system then downloads and installs all the packages that we specified along with the dependencies these might have.

4.1.2 Additional software

Another tool was used in this project: RStudio. This tool is not necessary for the final outcome. However, it made the process of analysis and plotting much more easier. If desired, RStudio can be downloaded from here (<https://www.rstudio.com>) and easily installed.

4.2 Formatting the data

Before we perform an analysis on the data, it must be normalized. As shown in the figure above, the data does not have a head that indicates us the name of every variable. Every time that any of the 500 tables is loaded into R, the software will provide standard names to those variables. For an easier work and better result, our implementation will rename the variables so that they match with the description provided by Google.

X6000000000	X9000000000	X3418309	X0	X4155527081	X0.001562	X0.06787
6e+08	9e+08	3418309	1	329150663	0.0015680	0.067870
6e+08	9e+08	3418314	0	3938719206	0.0003071	0.080440
6e+08	9e+08	3418314	1	351618647	0.0003004	0.080440
6e+08	9e+08	3418319	0	431052910	0.0004606	0.077150

Figure 4 Sample of data from the Google data set

As we can see in the figure above, R generated automatically names for the variables. However, these names mean nothing and they should be properly named. For instance, the first field in the picture is the Mean CPU field, but

instead we see X600000000. Because all of this, we use R to rename all the variables:

We import the data set first. In the final code this input is substituted for a piece of code that reads the input file as an argument passed to the script.

Import data from csv file	<code>data <- read.csv("part-00000-of-00500.csv")</code>
---------------------------	---

We rename the variables

Renaming columns	<code>colnames(data)[1] <- "StartTime"</code>
	<code>colnames(data)[2] <- "EndTime"</code>
	<code>colnames(data)[3] <- "JobId"</code>
	<code>colnames(data)[4] <- "TaskIndex"</code>
	<code>colnames(data)[5] <- "MachineId"</code>
	<code>colnames(data)[6] <- "MeanCPU"</code>
	<code>colnames(data)[7] <- "CanMemory"</code>
	<code>colnames(data)[8] <- "AssignedMem"</code>
	<code>colnames(data)[9] <- "UnmappedCache"</code>
	<code>colnames(data)[10] <- "TotPageCache"</code>
	<code>colnames(data)[11] <- "MaxMem"</code>
	<code>colnames(data)[12] <- "MeanIO"</code>
	<code>colnames(data)[13] <- "MeanDiskSpace"</code>
	<code>colnames(data)[14] <- "MaxCPU"</code>
	<code>colnames(data)[15] <- "MaxIOTime"</code>
	<code>colnames(data)[16] <- "CPI"</code>
	<code>colnames(data)[17] <- "MAI"</code>
	<code>colnames(data)[18] <- "Sample"</code>
	<code>colnames(data)[19] <- "AggregType"</code>
	<code>colnames(data)[20] <- "SampledCPU"</code>

Once we have renamed all the variables, the data is ready to be analyzed.

The final result after the normalization and variable discarding looks like this figure:

	MeanCPU	CanMemory	AssignedMem	UnmappedCache	TotPageCache	MaxMem	MeanIO	MeanDiskSpace	MaxCPU	MaxIOTime	CPI	MAI
413844	3.705e-02	9.338e-03	0.010600	2.308e-04	5.436e-04	0.0104400	7.153e-04	2.861e-06	0.0653100	4.388e-02	1.5670	0.0000000
1387326	4.482e-05	5.966e-03	0.009705	1.850e-04	4.435e-04	0.0059740	5.722e-06	0.000e+00	0.0002766	1.215e-03	17.2000	0.0782800
1473337	1.755e-04	1.135e-03	0.001934	3.099e-04	3.977e-04	0.0011350	1.907e-06	0.000e+00	0.0007372	7.153e-05	5.0070	0.0171600
2365208	5.292e-02	2.997e-02	0.031010	1.486e-03	1.593e-03	0.0306400	5.951e-04	1.411e-04	0.0745800	5.142e-03	2.0580	0.0033400
251461	1.804e-02	9.766e-03	0.011980	1.431e-05	3.242e-05	0.0100900	2.146e-04	2.861e-06	0.0758100	3.571e-03	1.2850	0.0017930
190673	9.186e-03	1.503e-02	0.015520	5.341e-05	5.817e-05	0.0151400	3.815e-06	2.861e-06	0.0633500	2.146e-04	2.4440	0.0061320
1479806	1.278e-04	1.637e-03	0.001940	1.040e-04	2.079e-04	0.0016400	1.049e-05	3.719e-05	0.0024410	1.572e-03	13.7500	0.0715300
783804	1.791e-02	6.189e-02	0.077270	5.447e-03	6.096e-03	0.0637200	4.768e-05	3.777e-04	0.0720200	2.857e-03	3.4770	0.0086800

Figure 5 Sample of the data set once normalized

Chapter 5

5. Data Analysis

In the last chapter we saw the aspect of the Task Resource Usage table. Besides that, we renamed the variables in order to make it friendlier.

In this chapter we perform a descriptive analysis of the different variables, then we discard several of them. Once we have the final set, we perform a Principal Component Analysis (PCA), a test and train procedure, a K-Means analysis and the same test and train procedure as before.

We finish this chapter with our interpretation of the results of the analysis.

5.1 Descriptive Analysis of the Variables

In the previous chapter we talked about the variables of the data set and we renamed them according to the data provided by Google. We describe here the meaning of these variables.

Variables 1 and 2: start and end time of the measurement period

These two variables represent the timestamp for every record, which is in microseconds since 600 seconds before beginning of the trace period, and recorded as a 64-bit integer. Besides these numbers, there are two special time values as well:

- A time of 0, that represents events that occurred before the beginning of the trace period.

- A time of $2^{63}-1$ that represents events that occur after the end of the trace period.

Variables 3, 4 and 5: Job ID, Task Index and Machine ID.

Every job is assigned a unique 64-bit identifier. These IDs are never reused. Task index is composed by the job ID of his job and a 0-based index within the job. Regarding the Machine ID, it is unique as well but it may stay the same when a machine is removed from the cluster and returned.

Variables 6 and 14: Mean and Maximum CPU

Both variables are measured in units of CPU-core seconds per second. Besides that, one represents the mean CPU usage and the other one represents maximum CPU usage.

Variables 7, 8, 9, 10 and 11: Memory related

All these variables are memory related. We must take into account that memory isolation is achieved through Linux containers, so some kernel memory usage on behalf of the tasks is accounted to the task. Here we describe the different variables:

- Canonical Memory: Memory usage measurement. Represents the number of user accessible pages.
- Assigned Memory: Memory usage based on the memory actually assigned to the container but not necessarily used.
- Unmapped Cache: represents Linux page cache not mapped into any user space process.
- Total Page Cache: Total Linux page cache.
- Maximum Memory: The maximum value of the canonical memory usage measurement observed over the measurement interval. This value might not be available for some tasks.

Variables 12, 13 and 15: Mean and Maximum Disk Input/Output time, Mean Disk Space

Disk I/O time is measured using subsystem of the Linux containers. Usage measurements are the sum across all disks on the machine, in units of disk-time seconds per second. Disk space represents runtime local disk capacity usage. Distributed storage is not accounted in this trace.

Variables 16 and 17: CPI and MAI

Cycles per Instruction and Memory Access per Instruction are collected from processor performance counters and all the machines do not collect it. Memory Access per Instruction measurements are based on last-level cache misses.

Variables 18, 19 and 20: Sample, Aggregation Type and Sampled CPU

Sample portion is the rate between the number of expected samples to the number of observed samples. Aggregation type defines the type of the aggregation depending on the Linux containers. Aggregation is related to the length of each measurement period. Sampled CPU represents the mean CPU usage during a random 1-second sample in the measurement period. This data can be used to build a stochastic model of CPU usage.

Distribution

Before continuing with the process, we analyzed the distribution of the variables.

The distribution of the variables is positively skewed. This means that it has a small variability and most of the values are within a small range of values. As an example, we offer details about two of the most relevant variables in terms of distribution.

MaxCPU

Positively Skewed Distribution. Standard Deviation: 0.10. Mean: 0.07

The figure below shows the data distribution of this variable.

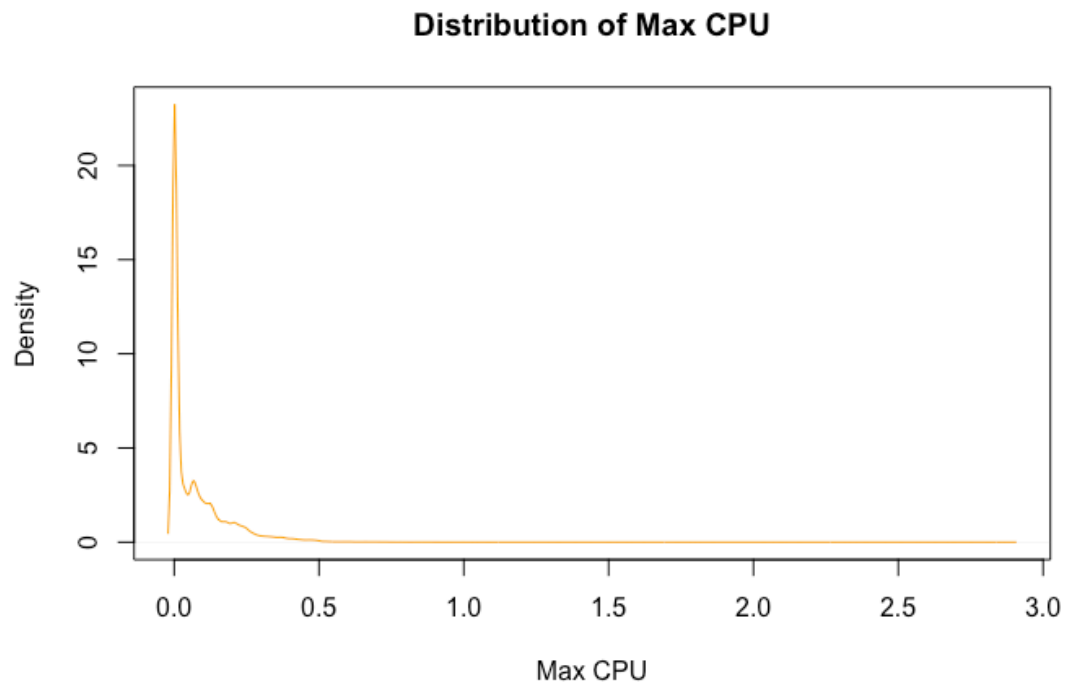


Figure 6 Max CPU distribution

Looking at the Standard Deviation, we see that the value is small, meaning that there is a small variability with respect to its mean.

Besides that, if we look at the plot of the data, it seems obvious that most of the data is between '0' and '0.5', as we can see in the plot below.

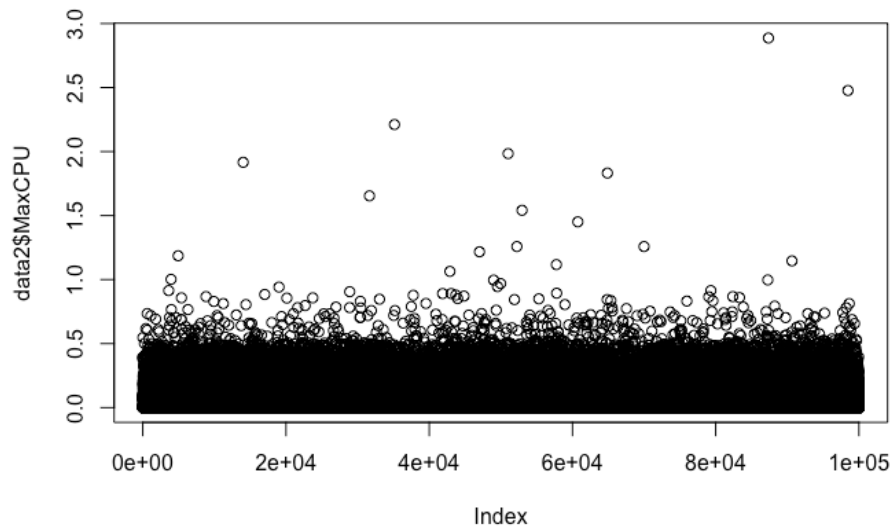


Figure 7 Max CPU distribution of data

MaxMem

Positively Skewed Distribution. Standard Deviation: 0.03. Mean: 0.019

Distribution of Max Memory

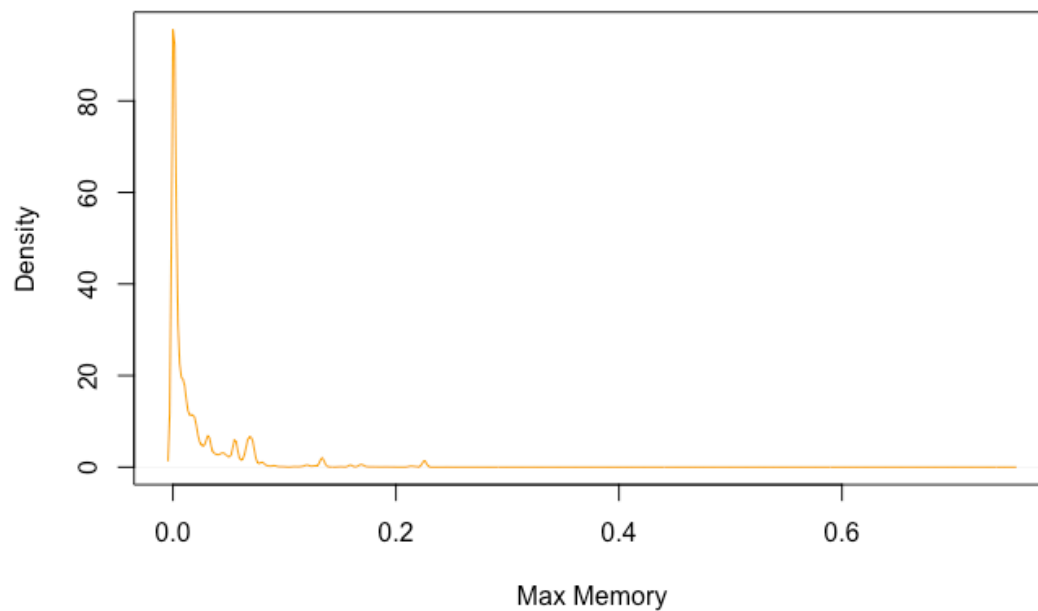


Figure 8 Max Memory distribution

As happened with the previous variable, standard deviation has a very low value, indicating that it exists a small variability with respect to the mean.

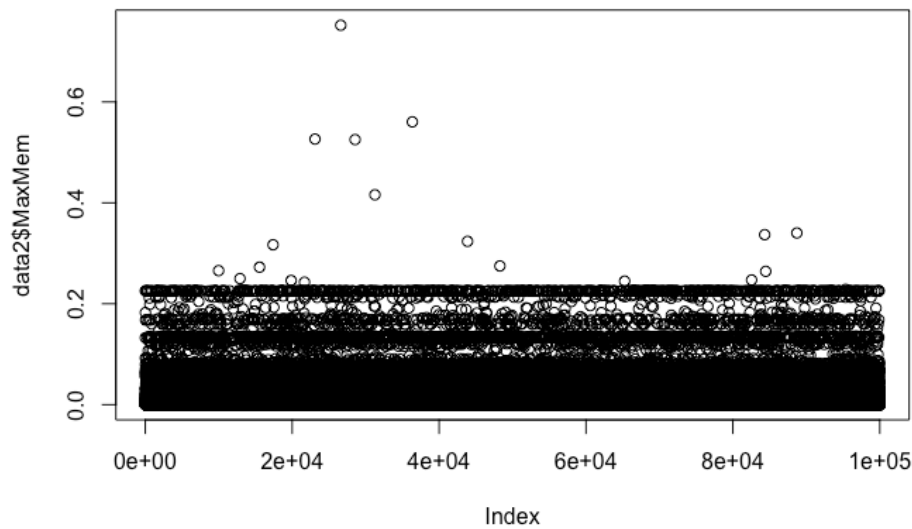


Figure 9 Max Memory distribution of data

Again, if we look at the plot of the data, most of the data is between '0' and '0.2', as we can see in the plot above.

Taking into account the distribution and the low variability of the different variables, we will be dealing with a lot of data that is very similar to each other. As a result, some of the plots will not be easy to interpret, making it difficult to draw a conclusion.

Summary

In the figure below we represent a summary of the variables previously explained.

Based on this information, we will proceed with the variable discarding, since not all the information seemed to be relevant.

Variable	Related to
1,2	Measurement Period. Time.
3,4,5	ID: Job ID, Machine ID, Task ID

6,14	CPU
7,8,9,10,11	Memory
12,13,15	Disk: space, Input/Output time
16,17	Instructions
18,19,20	Additional information like Sample measurements and others.

Figure 10 Variable comparison table

5.2 Variables Discarding

The aim of the Principal Component Analysis, PCA, is to reduce the dimension of the data set. That is, to discard variables by grouping them in different components that will represent the vast majority of the data. Before doing a PCA analysis, and due to the nature of the analysis, we have identified several variables that we will remove manually from the data set. These variables are: 1, 2, 3, 4, 5, 18, 19 and 20. We discarded these variables for not being appropriate or relevant for the PCA analysis.

For instance, variables 1 and 2 would have been useful if they followed a standard time format that on which we could apply a time series analysis for prediction. However, that was not the case and we proceed to discard them.

Variables 3, 4 and 5 are unique identifiers that could not provide additional value to the data set.

Variables 18, 19 and 20 are samples and aggregation types that would not be useful for our analysis since we analyzing a wider and more detailed data. As a consequence, there is no need for such sample variables.

At this moment, our data set is composed by 12 variables. In the next analysis we reduce the dimension of the data set to a small number of principal components and we explain the relationship between the components and the variable as well as the meaning of the results.

In our code, variable removal is accomplished by doing:

```
data$name_of_the_variable <- NULL
```

5.3 Principal Components Analysis (PCA)

5.3.1 Introduction

Principal components analysis is a statistical procedure that, given a set of possibly correlated variables, makes a conversion into linear uncorrelated variables called Principal Components.

When we gather data from a big data set, usually we want to take as many variables as possible. However, taking too many variables will lead to a high number of correlation coefficients. For example, if our data set has 20 variables, we should consider $\binom{20}{2} = 190$ possible correlation coefficients. The higher this number, the harder to visualize relationships between these variables.

Another issue that PCA tries to ease is the fact that if we have many variables, they would probably be highly related between them. This could mean that these variables are measuring the same data under different point of view.

In order to study the relationships between p correlated variables, the original data set can be converted into a new data set of uncorrelated variables called *set of principal components*.

These new variables are lineal combinations of the previous ones and they are composed according to the importance in terms of total variability of the original data set.

We pretend to get $m < p$ variables that would be linear combinations of the p original variables, representing most of the variability of the data.

5.3.2 Analysis

The table used for this study has about 2.5 million of rows. However, since working with such a large data set would be really difficult in terms of calculations and data management, we will work with a randomly selected sample of 100.000 registries.

This is the command used in R to obtain this new data set:

```
data2=data[sample(nrow(data), replace=F, size=100000),]
```

Once we have our final data set, data should be normalized. More specifically, we substituted any missing value with 0.

```
data2[is.na(data2)] <- 0
```

To perform the PCA analysis over this data set, we use *prcomp* function of R. According to the R documentation, there are two possible ways of doing a PCA analysis in R. One of them is *prcomp*, and the other one is *princomp*. R help specifies that *prcomp* uses *Singular Value Decomposition (SVD)*, and SVD has slightly better numerical accuracy, making *prcomp* the preferred function.

```
pca <- prcomp(data2, scale=TRUE)
```

Once the analysis finishes, a summary of the components can be displayed.

```
summary(pca)
```

```
Importance of components%s:
      PC1    PC2    PC3    PC4    PC5    PC6    PC7    PC8    PC9    PC10
Standard deviation  2.1782 1.4685 1.3606 1.1340 0.95316 0.70680 0.49182 0.42818 0.23574 0.21704
Proportion of Variance 0.3954 0.1797 0.1543 0.1072 0.07571 0.04163 0.02016 0.01528 0.00463 0.00393
Cumulative Proportion 0.3954 0.5751 0.7294 0.8365 0.91223 0.95386 0.97402 0.98930 0.99393 0.99786
      PC11    PC12
Standard deviation  0.14946 0.05822
Proportion of Variance 0.00186 0.00028
Cumulative Proportion 0.99972 1.00000
```

Figure 11 Summary of the PCA analysis with 12 components

As we can see in *Proportion of Variance*, the first four components represent the majority of the data. The first four components represent the 83.6 % of the total entries in the data set.

Below, we can see a bar plot of the importance of these components in the data set:

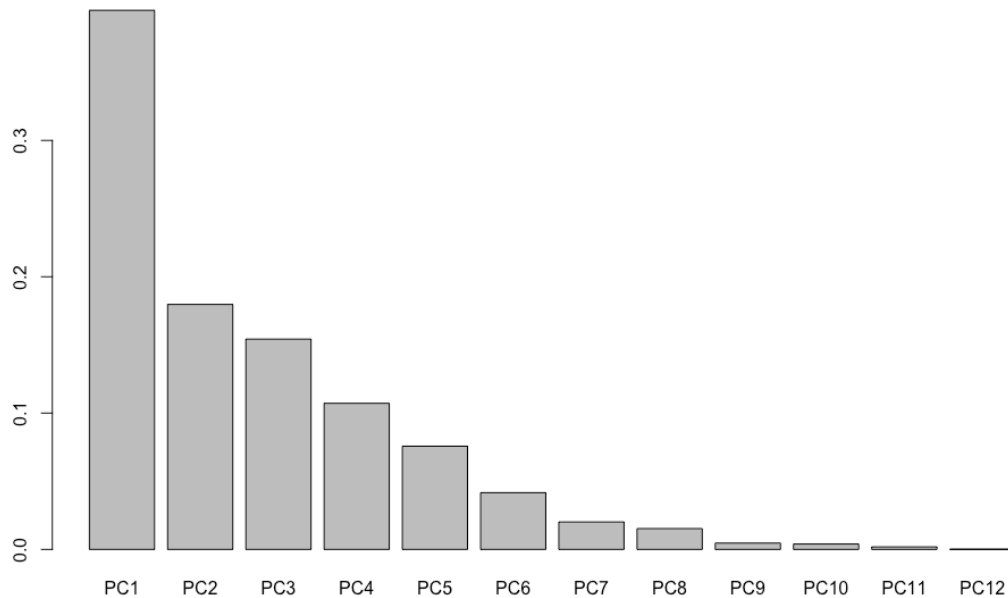


Figure 12 Bar plot of the proportion of the variance

This plot represents the importance of the components. It basically plots the *Proportion of variance*. As we mentioned above, this represents the weight of each component, or how much of the data represents every component.

Now that we have the components, it is interesting to see the relationship between these components and the original variables of the data set.

In the picture below we can clearly see that there are three groups of related variables. In the first place, we have CPI and MAI (both related to instructions of the code) directly related to Principal Component 1 (PC1). Another group of variables is the one composed by CPU and I/O time. These four variables are grouped around the same component PC2. The third group is the one composed by memory-related variables. Since they are located between the negative side of both PC1 and PC2, they have similar significance in both components, as we will see later.

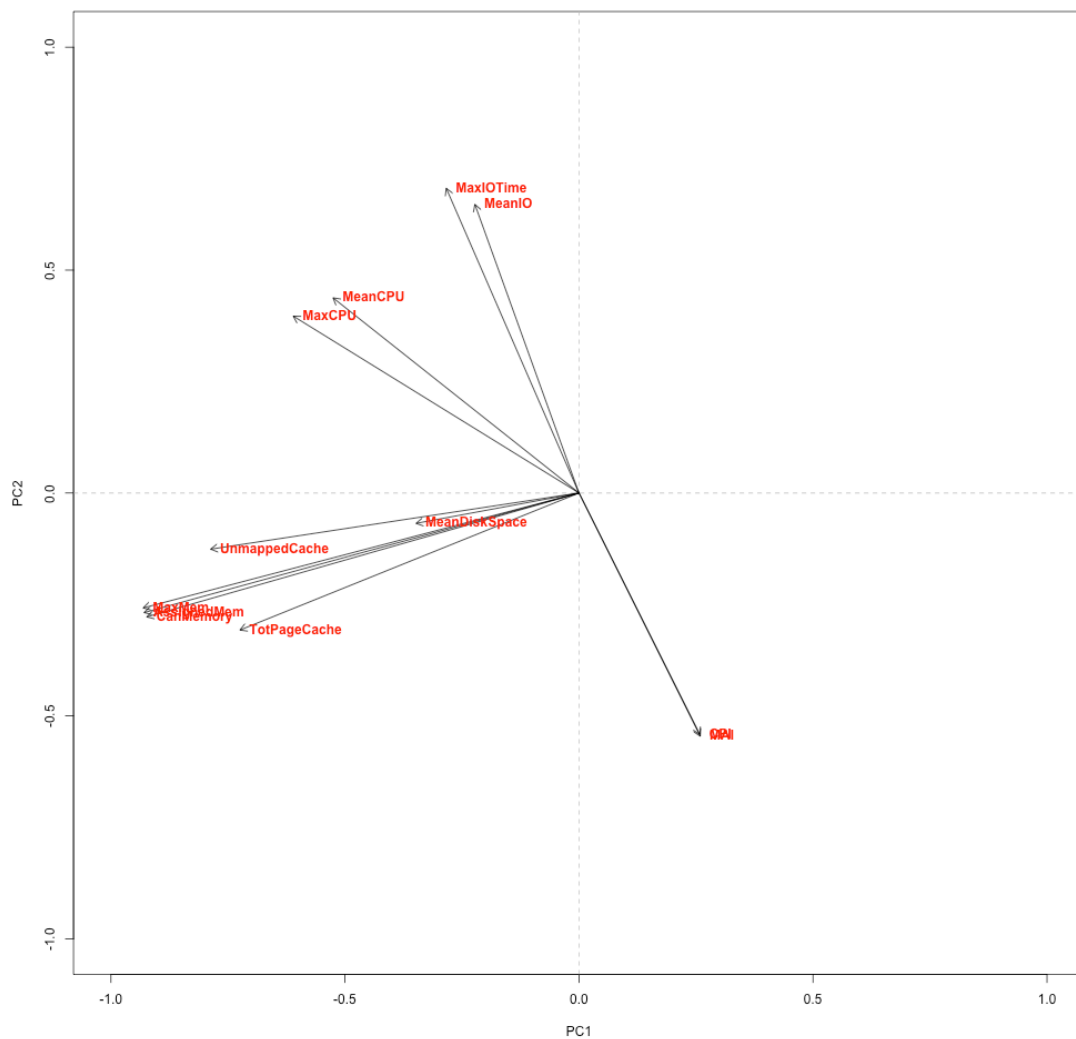


Figure 13 Correlation between variables and PC1 and PC2

As we saw in the previous plot, all memory related variables have a similar weight, being more close to PC2 than to PC1. However, other variables that are important in PC2 are not significant for PC1, such as those related to the CPU and storage.

Another important aspect that we can see also in the new plot below is the relationship of instruction related variables with PC1. Analyzing this plot we realized that there is a third group (PC3) that is not represented in the previous biplot.

As we can appreciate, CPI and MAI variables have an important weight on PC1. CPU and Input/Output-related variables have the same towards PC2. However, something that was missing before is shown here: a third group composed by cache, IO and instructions-related variables have a significant presence on PC3. These variables are: UnmappedCache, TopPageCache, MeanIO, MaxIOTime, CPI and MAI.

This relationship might mean that the cache used by these systems and jobs are associated with the storage devices. This also might mean that the cache is probably a software cache. According to Intel, higher CPI values mean that there is more latency in the system. Cache misses, Input/Output and/or other bottlenecks can cause this latency. According to this definition, our observation makes sense since this group of variables are all related to cache, Input/Output and CPI/MAI. According to the information on Figure 11, Principal Component 3 (PC3) represents 15% of the data. This means that 15% of the analyzed data could suffer from cache misses and probably I/O bottlenecks.

In the plots below, the darker grey represents PC1, the medium grey represents PC2 and the light grey represents PC3.

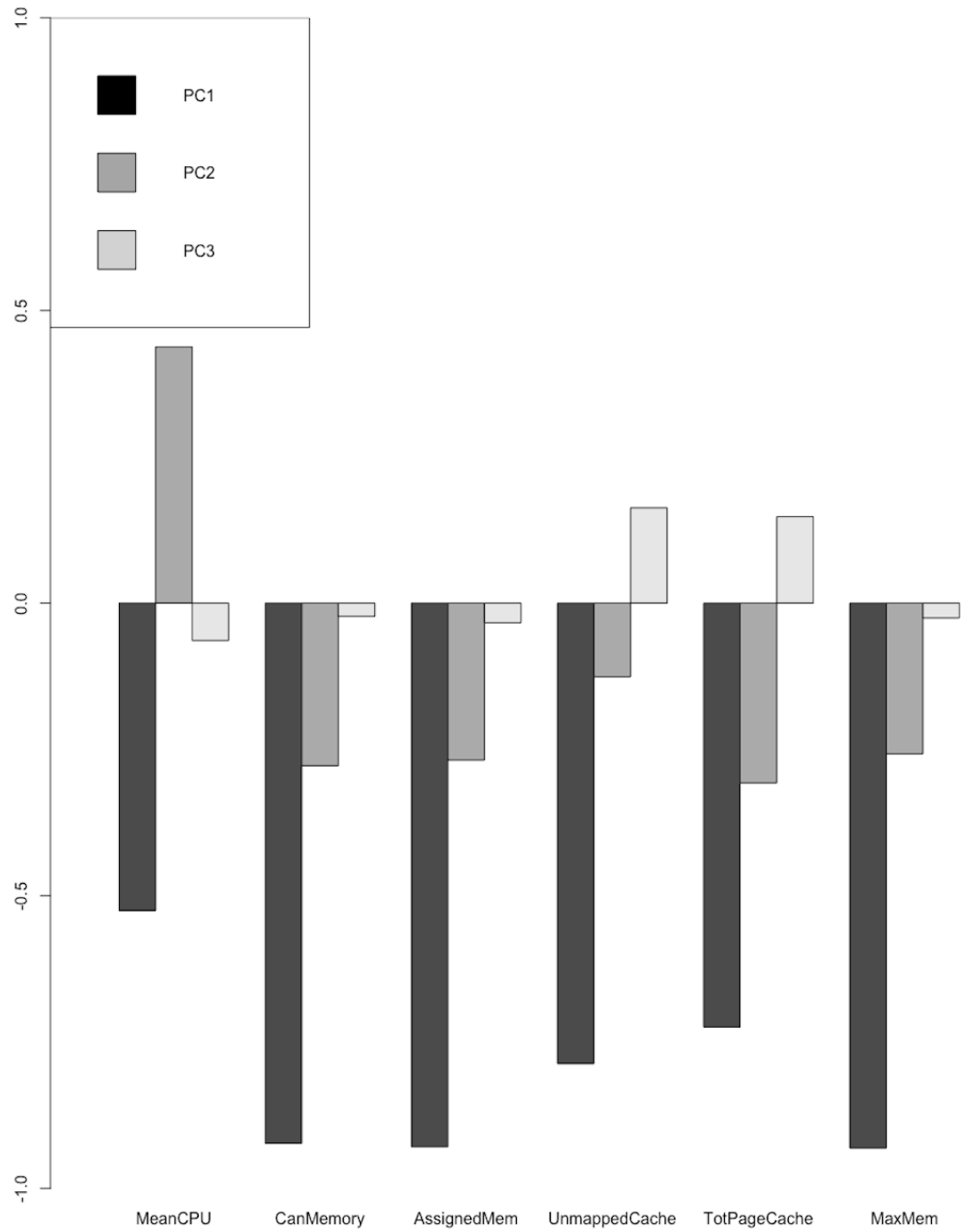


Figure 14 Bar plot of correlation between Principal Components and variables

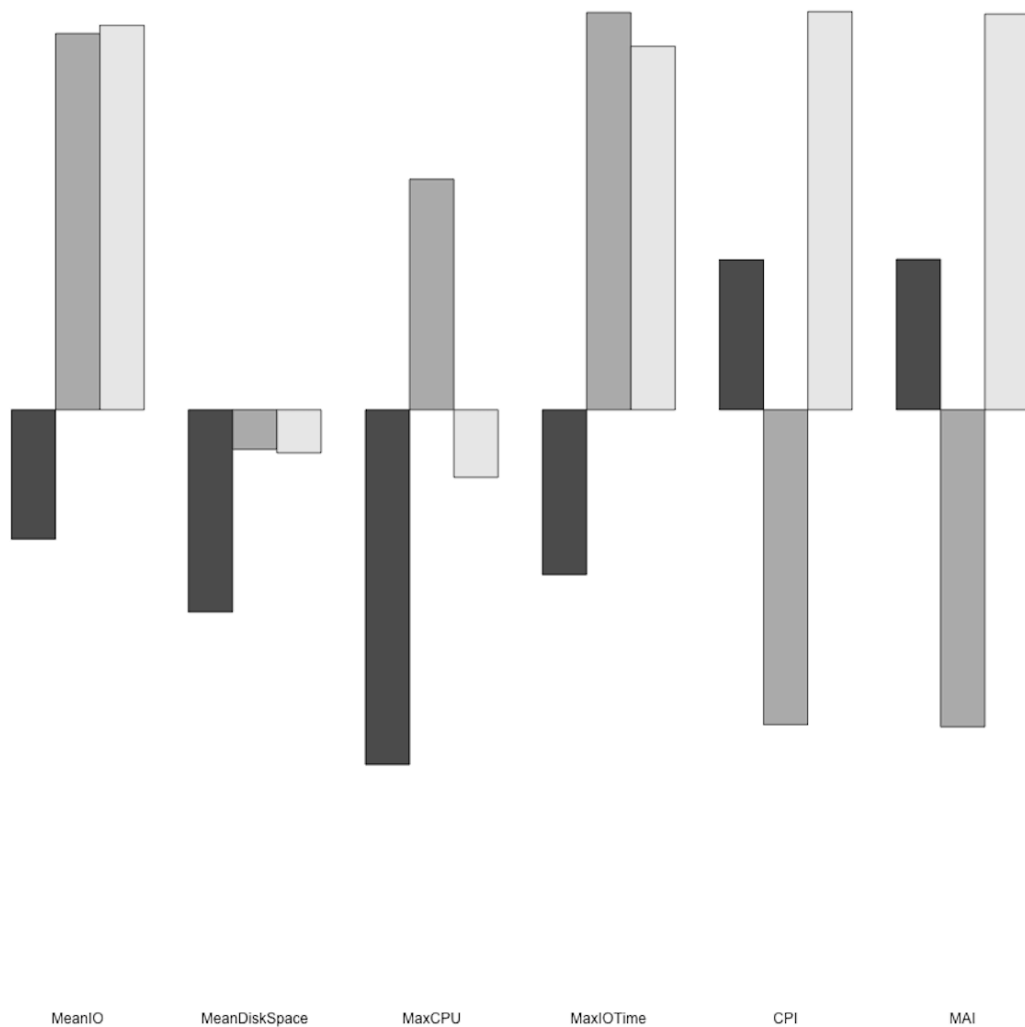


Figure 15 Bar plot of correlation between Principal Components and variables

We finish this analysis with an interesting plot that shows the relationship that a variable has with each other. In this plot, the red line represents the type of relationship between the variables. If the line is a perfect diagonal, the variables are linearly related. This could let us discard or group some of the variables, reducing even more the dimension of the data set.

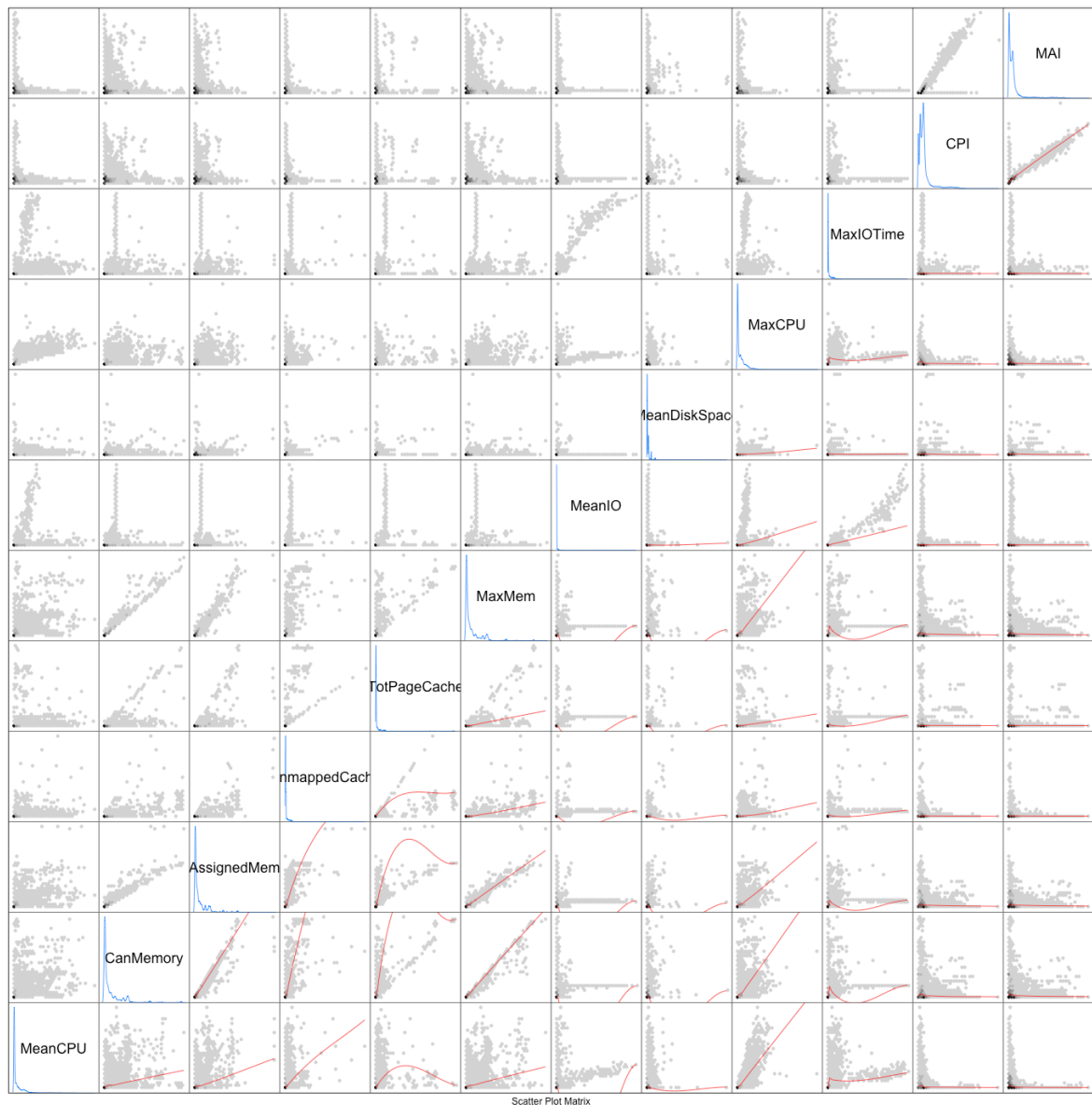


Figure 16 Variables relationship in a matrix

In this plot, we can avoid the upper left part of the matrix, since it represents the inverse of the lower right part of the matrix, which is where we painted the trend of each variable relationship.

There are some variables that feature an almost linear similarity. For instance, *Mean Memory and Assigned Memory* or *CPI* towards *MAI*. In the case of *CPI* and *MAI* variables, their relationship with all the rest of the variables is mainly the same. If we were trying to reduce even more the dimension of the data set, we could remove one of this two variables and the final result should not be significantly changed.

5.4 K-Means

Another tool to analyze data, such as the one we are working with, is K-Means. K-Means clustering is a method of vector quantification that is popular in data mining. It stores k centroids that it uses to define clusters. A point is considered to be in a particular cluster if it is closer to that cluster's centroid than any other centroid.

In this project we use K-Means along with a test and train procedure in order to get a predictable categorization of the data.

5.4.1 Introduction

In order to perform a K-means analysis with R, we decided to use the function 'kcca', from the library 'Flexclust'. This function perform a k -centroids clustering on a data matrix. Within this function we chose the 'family' parameter to be 'kmeans', which is based in an Euclidean distance and mean. Besides that, we used kmeans++ (kmeanspp) centroid initialization.

We performed a K-Means analysis with a $k=3$. We choose this number because $k=2$ would be very simple and beyond 3 we get too many partitions of similar data.

In the first place, we decided to use 94% of the data for training purposes and the rest 6% of the data for testing or guessing purposes. More specifically, we took the first 8.000 rows of the data set for the training part of the process. This represents 94% of the 8.500 rows selected. The rest, from row 8.001 to 8.500 represents our testing and learning data. The reason of choosing just 8.500 tasks to work with is that plotting 100.000 dots would be a waste of resources, it would take much more time to process and it would difficult reading and interpreting the resulting image.

We used then a PCA analysis in order to use just two Principal Components instead of working with 12 variables, which would difficult the K-means analysis.

In the figure below we can see how the data set is partitioned in 3 clusters. The numbers represent the centroids of every cluster, and the line width between them represents how much these clusters are related. For instance, clusters 1 and 3 have a stronger relationship than cluster 2 and 3.

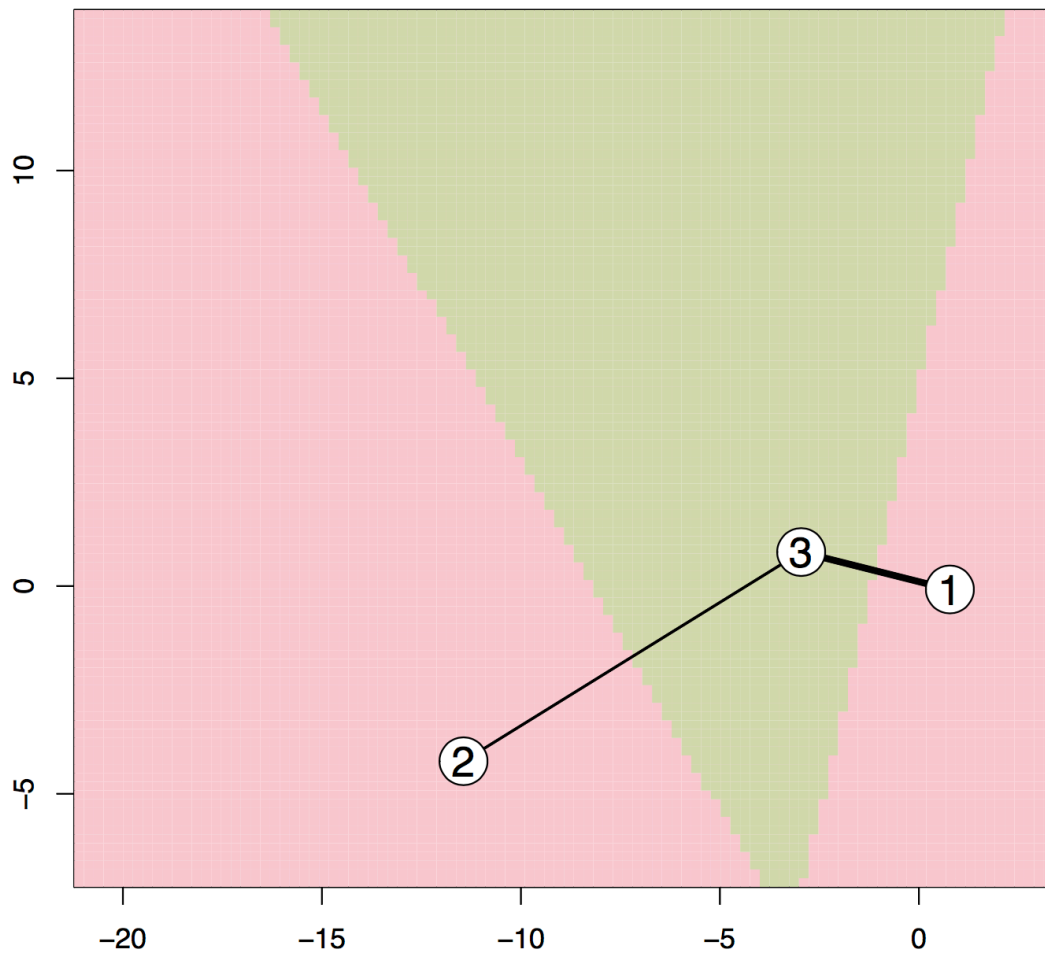


Figure 17 Clusters partitioning using K-means

As we mentioned above, since plotting 100.000 points would render useless due to the massive number of points in the image, we decided to plot the first 8000 rows in form of points that looks like the image below:

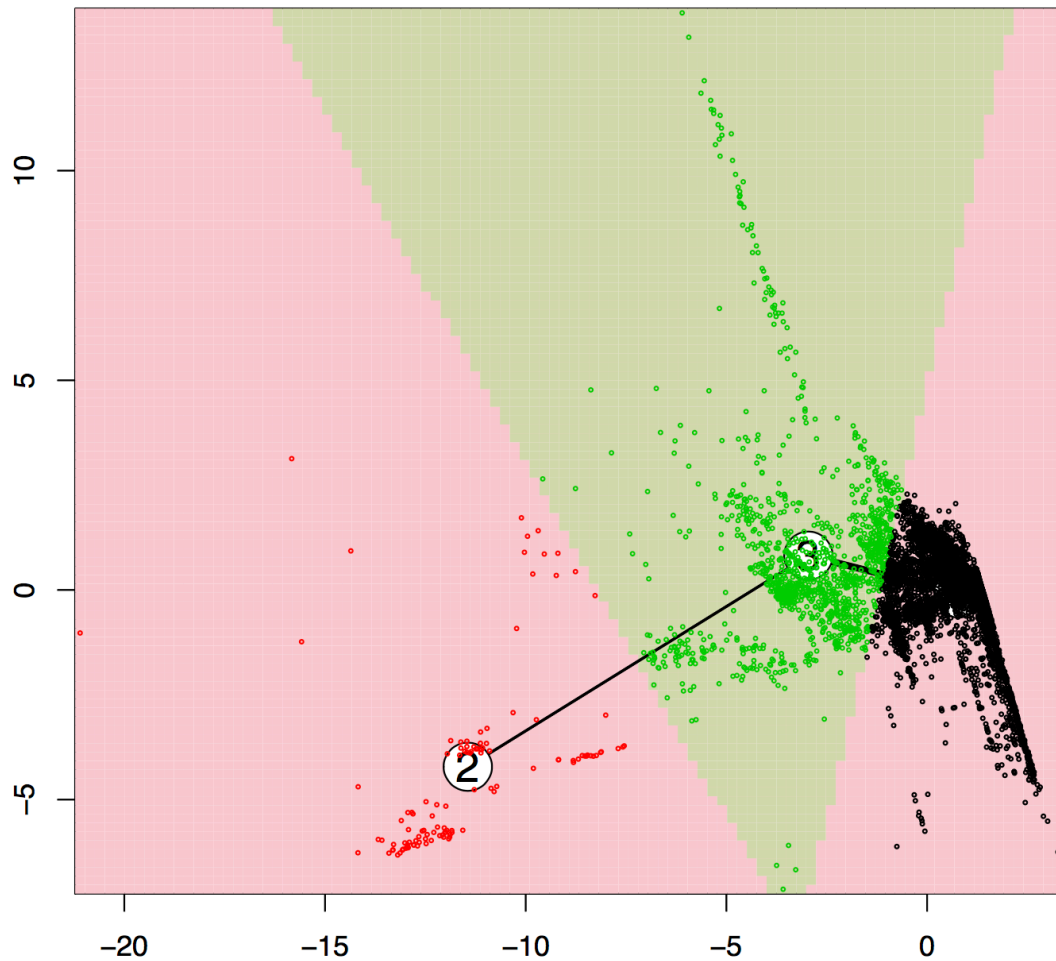


Figure 18 Data representation along the clusters

This plot represents the three different clusters or groups and the points related to them.

Looking at Figure 18, we can observe that apparently cluster one (black) has a bigger density of points followed by cluster three (green). Cluster two (red) is the smallest one, meaning that it is the one with the fewest entries.

5.4.2 K-Means Test and Train

The idea behind testing and training is that, given a data, the algorithm learns from it and it is able to predict how the next data that we analyze would be

categorized. In our case, we provided 500 extra rows to see where they would fit.

For this purpose, we used the function 'predict', a generic function for predictions from the results of various fitting models. In this case, we used 'predict' with data from the cluster we created before and also with data from the cluster and the two principal components of the PCA mentioned before.

The result is in the figure below:

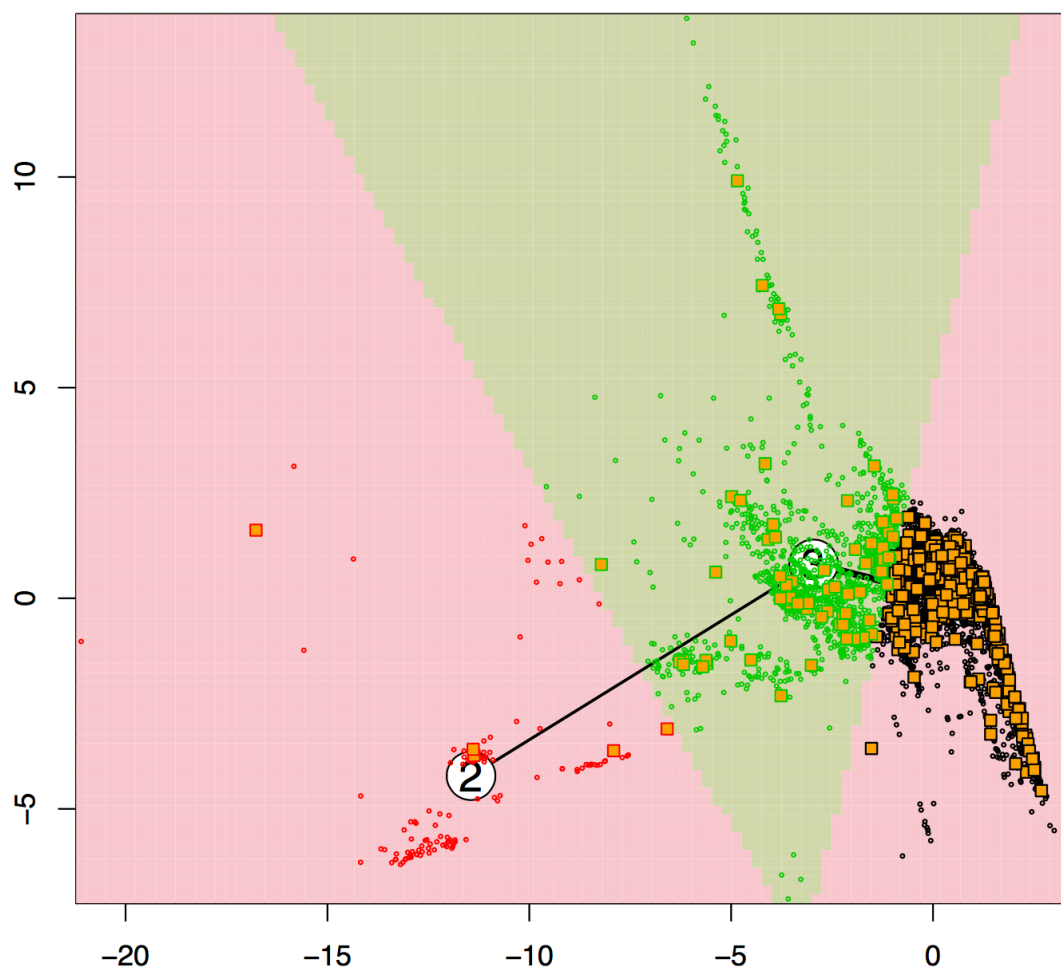


Figure 19 Training test data

As we can see in the figure, most of the new entries would be fit into the cluster on the right side.

In the next section we will interpret these results more in detail.

5.5 Interpretation

The objective during the analysis was to use statistical methods for analyzing large data. In order to achieve a significant result, we reduced the dimension of the data set by using the PCA analysis. Then, we performed a clustering analysis in order to group observations. Finally, we performed a small prediction based on the clustering results.

In the first place, we will come to a conclusion based on the PCA analysis. As we mentioned before, principal components represent 100% of the data set. However, even with a non-normal distribution data set like the one we are studying, a few components represent most of the variance of the data set. In our case, the first three components represented 72% of the data set. Since the data set was randomly composed from a bigger data set, we can consider that our analysis results are valid for most of this data set composed by 2.5 million of results.

Based on the results obtained in section 5.3, we determined that all the data in the data set is represented by three groups of variables. As we explained before, this data is made of tasks, meaning that the tasks sent to the clusters in order to be executed made an extensive use of CPU, memory and input/output operations. We stated that CPU and input/output time were grouped under the same component PC2. However, this does not necessarily mean that they are related. In fact, a further inspection shows that there is no high processor usage during the input/output operations.

The picture below shows this behavior:

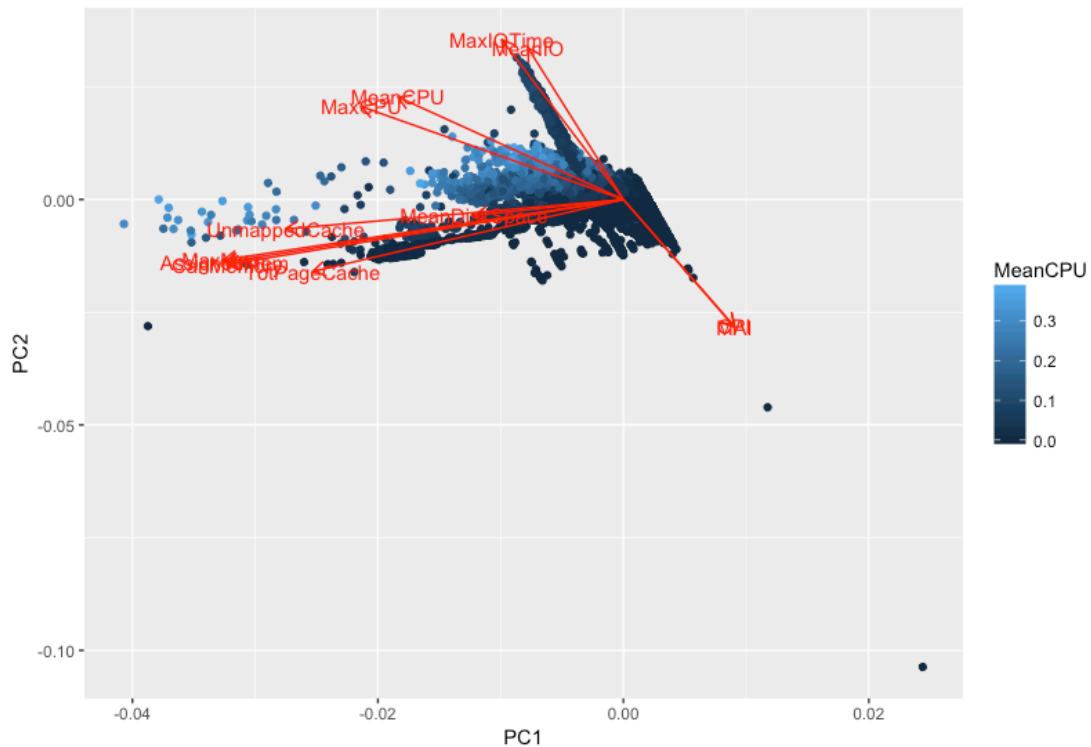


Figure 20 Mean CPU usage across the different variables

As we can see in the figure, a higher processor usage is present in those tasks related to the memory but not in those related to input/output operations. According to what we explained before about the reason behind the high CPI (cycles per instruction) and MAI (memory accesses per instruction) values, a normal or even low processor usage in these tasks is reasonable, since they probably have a bottleneck problem somewhere else. We do know though that the processor is not causing this issue.

Another interesting aspect of the data is that principal component 3, PC3, showed this relationship too. The variables related to this component were those related to the cache, input/output time and CPI/MAI.

To summarize, while using PCA to analyze data we should consider more than two components, since a different relationship could emerge. In our case, PC3 demonstrated that there was a third group of variables not related to intensive processor usage, despite the output of the biplot of PC1 and PC2 where CPU-related variables seemed to be aligned with input/output operations.

Regarding the K-means study, we decided to use three clusters to represent the data. This clustering methodology allowed us to group data and also to predict new tasks and catalog them in a group.

We saw that most of the new tasks would be classified inside cluster number one. However, we were not aware of what different clusters meant. We created a new plot showing the relationship between the three clusters and the two first principal components of the PCA analysis. The result can be seen in this plot:

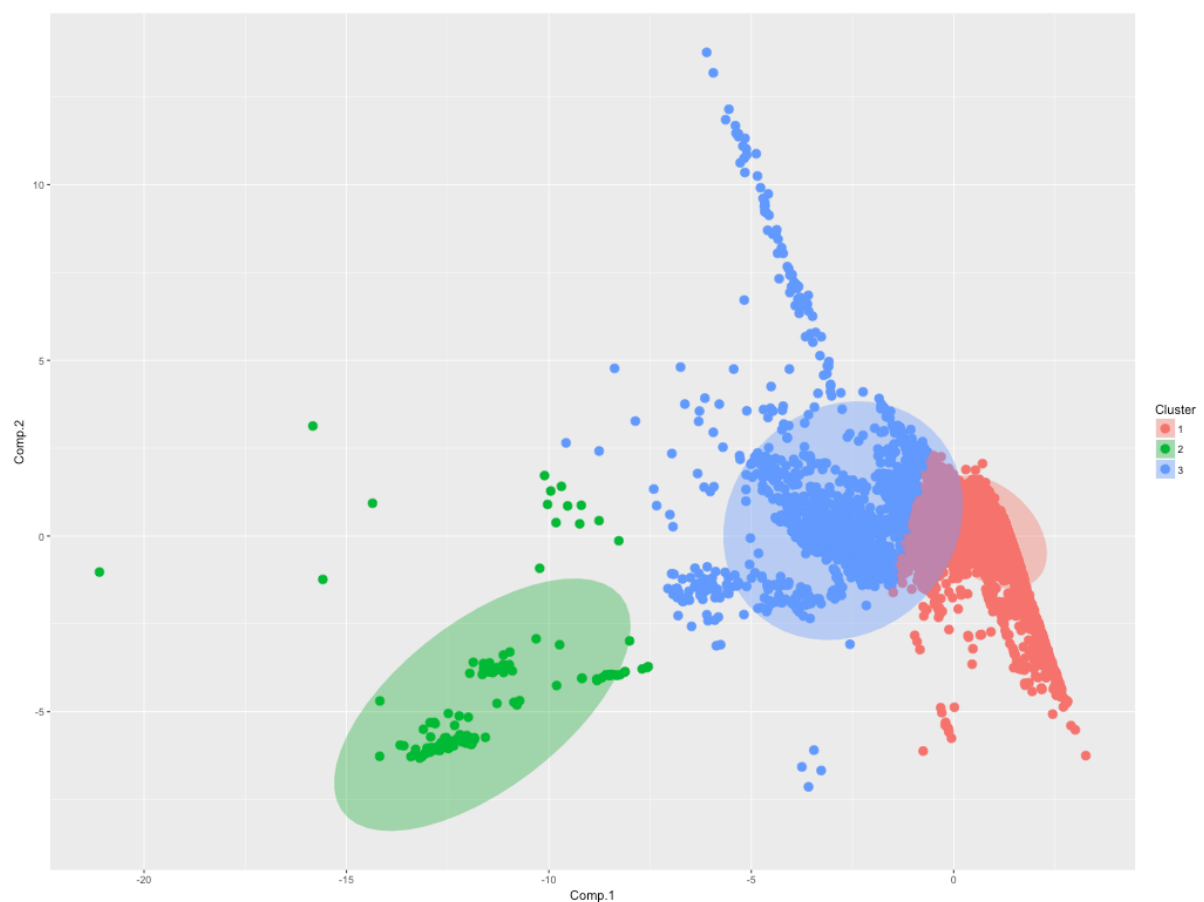


Figure 21 Data representation along the clusters with predictive ellipses and components

A further analysis of this relationship was necessary. According to the plot above, the data located in cluster 1 and cluster 3 should be related to principal component 1. After checking the data itself, we realized that this is indeed the relationship that we were seeing in the plot, as we can see in the figure below.

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6	Comp.7	Comp.8	Comp.9	Comp.10	Comp.11	Comp.12	Cluster
2237525	3.262462	-6.249467	-6.924043	4.120729	-1.1047292	-0.3433704	-0.142531385	-0.26437828	-0.012482175	-0.0070275804	0.081160177	8.832094e-03	1

Comp.2	Comp.3	Comp.4	Comp.5	Comp.6	Comp.7	Comp.8	Comp.9	Comp.10	Comp.11	Comp.12	Cluster
13.764036	-16.11536122	-4.1927028	0.63765079	2.76667725	0.9144433296	0.868907541	0.317555068	4.27603749	-0.0581171819	-0.1998753275	3

Figure 22 Data representing the Component 1 and 2 values with respect to Clusters

This figure shows that positive and high numbers of Principal Component 1 and Principal Component 2 are grouped into cluster one and cluster three respectively. As we saw before, Principal Component 1 groups CPI and MAI variables, while Principal Component 2 groups CPU and input/output variables.

In section 5.4.2 we established that most of the new tasks would fit in cluster one. Since cluster one has a strong relationship with Principal Component 1, we could summarize the outcome of this analysis by stating that many of the new tasks sent to these clusters would have significant values of CPI and MAI.

However, we should keep in mind what we mentioned in section 3.1. According to the documentation released by Google, they detected a possible error in CPI and MAI measurements, probably due to the system they use for capturing the data.

Chapter 6

6. Testing

We have tested the solution with different traces from the same data set, always using the *Task Resource Usage* table. It is important to note that, since the data required manual manipulation, it will not work with a different table or different trace.

In order to reproduce the tests, we can use Rscript, a command line tool included with R, to execute the framework:

```
$ Rscript pca_google.R <input_file>
```

where <input_file> should be the file provided by Google in comma separated values format. These files follow the following name format:

part-00000-of-00500.csv

Due to the size of the file, it cannot be included in the final project submission. However, we uploaded this data set here:

<https://github.com/e9169/UOCTfg>

The whole data set in which this work has been based on, can be downloaded following the instructions in this document:

https://drive.google.com/file/d/0B5g07T_gRDg9Z0lsSTEtTWtpOW8/view

6.1 Example

In this section we will run a complete example with a different data set. This additional data set can also be downloaded from:

<https://github.com/e9169/UOCTfg>

After downloading and uncompressing the data set, we executed the platform:

```
Rscript pca_google.R part-00001-of-00500.csv
```

The framework that we have developed will normalize the data accordingly and will perform all the analysis as well as create the different plots that help to understand the analysis. After a few minutes, a 'figures' folder is created inside the folder from which the code is executed. Inside this folder we put all the plots mentioned before.

The code might throw some 'null device' errors. This is because of the way we save 'png' images into the 'figures' folder. They can be discarded since they do not affect the performance or final result of the analysis.

Chapter 7

7. Conclusion and Future Work

In this final chapter we present the conclusions of the analysis. Besides that, we mention several topics that could be treated in the future.

7.1 Conclusions

With the help of the analysis of the data and the plots, we can render some interesting conclusions about the data type of the clusters used by Google. It will also help us to understand what is more relevant and, thus, propose a standard with fewer variables.

Starting with the PCA analysis, we concluded in section 5.5 that three components represented 72% of our data. It is important to mention that all the analysis performed in this study was done against a randomly selected data of 100.000 tasks. The original file used for this selection has about 2.5 million of tasks. This means that, if we do not use the whole data set of 2.5 tasks, we will get different results every time we use this solution. Working with such a number of tasks can be tedious and slow in performance, and using just the 100.000 first tasks of the data set would not take into account changes in the rest of the data set. That is, the first 100.000 tasks will not represent the whole data set. However, there is a higher change that 100.000 random tasks will.

In section 5 we concluded that CPU and Input/output variables were grouped under the same Principal Component. However, we also realized that this does not mean they are directly related, as we demonstrated. It is important then not to use just a simple analysis of a few principal components.

Besides that, we identified a potential problem with 15% of the tasks. This problem is related to possible bottlenecks due to input/output operations, cache misses or other problems.

Regarding the K-means analysis, we classified data into three groups and performed a prediction of several new tasks in order to see where they would fit. The conclusion was that most of the new tasks would be classified or grouped under the cluster number one. We also realized that cluster one is directly related with higher values of Principal Component 1, meaning that most of the new tasks would have a high value of CPI (cycles per instruction) and MAI (memory accesses per instruction).

We also noticed that higher values of mean CPU usage are related to the memory variables, meaning that using more processor power involves using more memory, something that we can consider a normal and expected behavior.

While trying to define a standard workload format that could integrate the most relevant data, we identified three areas with the PCA analysis. These three areas are: processor related variables, memory related variables and instructions related data. Another group of variables that should be part of this standard that we are trying to define is the one composed by time related variables, such as starting and ending of a job or task. However, this data was not part of the data set we analyzed, since the time variables provided were relative to a certain moment that was not properly specified.

Time-series data could help us by predicting in a different way, allowing us to compare the current K-means based prediction with this other method.

To summarize, we identified a potential problem with an important number of tasks, we grouped most of the data into three different principal components, we analyzed the relationships between these components and the variables of the data set, we grouped the data into clusters, analyzed the relationship between the clusters and principal components and we finally predicted the classification of new tasks by a train and test method on the k-means analysis.

7.2 Future Work

Since at the moment of the creation of this project we are not aware of an updated and commonly used format, this work could be highly improved in the future by implementing such a non-existent standard. This way, the analysis process could be fully automated.

In the previous section we have proposed a standard format based on the results of the Google workload traces. However, an exhaustive and dedicated analysis should be applied to many traces, since different traces represent different needs in terms of computing resources. Because of this, we can consider this work as a first and necessary step before we can consider a standard.

Besides that, additional analysis methods could be used, for example topological data analysis (TDA). TDA provides a framework to analyze high-dimensional, incomplete and noisy data sets. It provides dimensionality

reduction and robustness. The motivation of this kind of analysis is to study the shape of data. The main tool of TDA is persistent homology. These algorithms look for topological invariants across various scales of a topological manifold.

Another aspect that can be extended is prediction. As we explained before, a brief k-means based prediction is achieved with this framework. However, we could expect a more detailed approach by using decision trees, also known as regression trees, as a predictive model. This model is used in statistics, data mining and machine learning. The algorithm finds the variable that does the best job of separating the data into two groups. The operation is repeated with the other variables, resulting in a tree graph where each split represents a decision. The biggest advantage of this model is that it is really intuitive and can be easily understood by people with no or little experience.

We would like to mention an interesting library for prediction based on time-series that could be applied to a data set similar to the one this project is based on. The library is called Prophet, a Facebook open-source library for forecasting. As we mentioned before, due to the nature of the time-related variables of the data set of this project, we could not use this library for prediction, but it seems to be interesting and we should consider its application.

Basically, improving three major areas could extend this work: automation, wider data analysis and prediction.

8. References

Cs.ucsb.edu. (2015). *University of California Santa Barbara*. [online] Available at: <https://www.cs.ucsb.edu/~rich/workload/> [Accessed 28 May 2017].

En.wikipedia.org. (2017). *Principal component analysis*. [online] Available at: https://en.wikipedia.org/wiki/Principal_component_analysis [Accessed 28 May 2017].

Ftp.pdl.cmu.edu. (2013). *OpenCloud Hadoop cluster trace : format and schema*. [online] Available at: <http://ftp.pdl.cmu.edu/pub/datasets/hla/dataset.html> [Accessed 28 May 2017].

GitHub. (2012). *SWIMProjectUCB/SWIM*. [online] Available at: <https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository> [Accessed 28 May 2017].

GitHub. (2017). *google/cluster-data*. [online] Available at: https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md [Accessed 28 May 2017].

Klusacek, D. (2017). *Dalibor Klusacek - Home page*. [online] Fi.muni.cz. Available at: <http://www.fi.muni.cz/~xklusac/index.php?page=meta2009> [Accessed 28 May 2017].

Marín Diazaraque, J. (2017). *Análisis de Componentes Principales*. [online] Available at: <http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/AMult/tema3am.pdf> [Accessed 28 May 2017].

Planspace.org. (2013). *PCA, 3D Visualization, and Clustering in R*. [online] Available at: <http://planspace.org/2013/02/03/pca-3d-visualization-and-clustering-in-r/> [Accessed 28 May 2017].

Pucher, A. (2015). *Cloud Traces and Production Workloads for Your Research | Alexander Pucher*. [online] Alexpucher.com. Available at: <https://alexpucher.com/blog/2015/06/29/cloud-traces-and-production-workloads-for-your-research/> [Accessed 28 May 2017].

R-bloggers. (2017). *Passing arguments to an R script from command lines*. [online] Available at: <https://www.r-bloggers.com/passing-arguments-to-an-r-script-from-command-lines/> [Accessed 28 May 2017].

R-project.org. (2017). *R: The R Project for Statistical Computing*. [online] Available at: <https://www.r-project.org> [Accessed 28 May 2017].

RStudio. (2017). *RStudio*. [online] Available at: <https://www.rstudio.com/products/rstudio/> [Accessed 28 May 2017].

Rstudio-pubs-static.s3.amazonaws.com. (2017). *Análisis de Componentes Principales en R*. [online] Available at: http://rstudio-pubs-static.s3.amazonaws.com/2387_ee73f69e5ba04498a93e5bb12f636591.html [Accessed 28 May 2017].

Sthda.com. (2017). *Principal component analysis in R : prcomp() vs. princomp() - R software and data mining - Easy Guides - Wiki - STHDA*. [online] Available at: <http://www.sthda.com/english/wiki/principal->

component-analysis-in-r-prcomp-vs-princomp-r-software-and-data-mining
[Accessed 28 May 2017].

Terrádez Gurrea, M. (2017). *Análisis de Componentes Principales*. [online]
Available at:
https://www.uoc.edu/in3/emath/docs/Componentes_principales.pdf
[Accessed 28 May 2017].

Thiago G. Martins. (2017). *Computing and visualizing PCA in R*. [online]
Available at: <https://tgmstat.wordpress.com/2013/11/28/computing-and-visualizing-pca-in-r/> [Accessed 28 May 2017].

Tsafrir, D. (2015). *Parallel Workloads Archive: Logs*. [online] Cs.huji.ac.il.
Available at: <http://www.cs.huji.ac.il/labs/parallel/workload/logs.html>
[Accessed 28 May 2017].

Kodali, T. (2015). *Using Decision Trees to Predict Infant Birth Weights*. [online]
Available at: <https://www.r-bloggers.com/using-decision-trees-to-predict-infant-birth-weights/> [Accessed 21 June 2017].

En.wikipedia.org. (2017). *Big Data*. [online] Available at:
https://en.wikipedia.org/wiki/Big_data [Accessed 22 June 2017].

En.wikipedia.org. (2017). *Supercomputer*. [online] Available at:
<https://en.wikipedia.org/wiki/Supercomputer> [Accessed 22 June 2017].

En.wikipedia.org. (2017). *Topological Data Analysis*. [online] Available at:
https://en.wikipedia.org/wiki/Topological_data_analysis [Accessed 23 June 2017].

Software.intel.com. (2017). *Clockticks per Instructions Retired*. [online]
Available at: <https://software.intel.com/en-us/node/544403> [Accessed 24 June 2017].

Carmona, F. (2014). *Un ejemplo de ACP paso a paso*. [online] Available at:
<http://www.ub.edu/stat/docencia/Mates/ejemploACP.PDF> [Accessed 23 June 2017]

Facebookincubator.github.io. (2017). *Prophet*. [online] Available at:
<https://facebookincubator.github.io/prophet/> [Accessed 5 June 2017]

9. Annex A

In this chapter we provide the source code of the framework in R language. Besides that, we provide a public repository for the data sets related to this project.

Data Sets

<https://github.com/e9169/UOCTfg>

R code

```
# Installation requirements

require(solaR)
require(hexbin)
require(ggplot2)

if (file.exists("figures")){
  message ("Figures folder already exists")
} else {
  dir.create("figures")
}

# Read data (csv file)
args <- commandArgs(TRUE)
data <- read.csv(args[1])

# Removing unwanted columns manually
data <- data[ -c(1:5)]
data <- data[ -c(13:15)]

# Renaming column names
colnames(data)[1] <- "MeanCPU"
colnames(data)[2] <- "CanMemory"
colnames(data)[3] <- "AssignedMem"
colnames(data)[4] <- "UnmappedCache"
colnames(data)[5] <- "TotPageCache"
colnames(data)[6] <- "MaxMem"
colnames(data)[7] <- "MeanIO"
colnames(data)[8] <- "MeanDiskSpace"
colnames(data)[9] <- "MaxCPU"
colnames(data)[10] <- "MaxIOTime"
```



```

colnames(data)[11] <- "CPI"
colnames(data)[12] <- "MAI"

# I generate a random sample of the original dataset of
2.5 million rows.
data2=data[sample(nrow(data), replace=F, size=100000),]

# I normalize the data so there is no NA. I replace it
with 0 so that everything is numerical.
data2[is.na(data2)] <- 0

# PCA Analysis
pca <- prcomp(data2, scale=TRUE)

# With covariance matrix
(pca.cov <- prcomp(data2))
diag(1/sqrt(diag(cov(data2)))) %*% pca.cov$rotation %*%
diag(pca.cov$sdev)
(corvar2 <- pca.cov$rotation %*% diag(pca.cov$sdev))

# Summary
summary(pca)

# Correlation between variables and components
(corvar <- pca$rotation %*% diag(pca$sdev))

# Representation
# Bar plot of the weight of each component

png(filename="figures/barplot1.png", width = 1024, height
= 768)
barplot(summary(pca)$importance[2,])
dev.off()

# Correlation between variables and components PC1 and
PC2
png(filename="figures/correlation.png", width = 1000,
height = 1000)
plot(-1:1, -1:1, type='n', xlab='PC1', ylab='PC2', xlim =
c(-1,1))
abline(h=0, v=0, lty=2, col=8)
arrows(0, 0, corvar[,1], corvar[,2], length=.1)
text(corvar[,1], corvar[,2], colnames(data2), pos=4,
offset=.6, col=2, font=2)
dev.off()

# Correlation between variables and components PC1, PC2
and PC3 in a bar plot

```

```

png(filename="figures/barplot2.png", width = 1920, height
= 1080)
barplot(t(corvar[,1:3]),beside=TRUE, ylim=c(-1,1))
legend("topleft",legend = c("PC1","PC2","PC3"), fill =
c("black","darkgray","lightgray"))
dev.off()

```

```

# Variables relationship. Requires Hexbin and solar
png(filename="figures/matrix.png", width = 1400, height =
1400)
splom(data2[1:15000,1:12],
      panel=panel.hexbinplot,
      diag.panel = function(x, ...){
        yrng <- current.panel.limits()$ylim
        d <- density(x, na.rm=TRUE)
        d$y <- with(d, yrng[1] + 0.95 * diff(yrng) * y /
max(y) )
        panel.lines(d)
        diag.panel.splom(x, ...)
      },
      lower.panel = function(x, y, ...){
        panel.hexbinplot(x, y, ...)
        panel.loess(x, y, ..., col = 'red')
      },
      pscale=0, varname.cex=1.5
)
dev.off()

```

```

# Training and Test. K-Means.
library(flexclust)

```

```

Train <- data2[1:8000,]
Test <- data2[8001:8500,]

pca.train <- princomp(Train,cor=TRUE)
pca3.train <- prcomp(Train, scale=TRUE)
png(filename="figures/pca.train.png", width = 1024,
height = 768)
plot(pca.train)
dev.off()

```

```

pc1.train <- pca.train$scores[,1]
pc2.train <- pca.train$scores[,2]

```

```

X.train <- cbind(pc1.train,pc2.train)

```

```

pca.test <- predict(pca.train, newdata=Test)

```

```

pc.comp.test <- pca.test[,1:2]
pc1.test <- pc.comp.test[,1]
pc2.test <- pc.comp.test[,2]

X.test <- cbind(pc1.test,pc2.test)

cl <- kcca(X.train,k=3,kccaFamily("kmeans"),control =
list(initcent="kmeanspp"))

pred_train <- predict(cl)
pred_test <- predict(cl,newdata=X.test)

pdf(file="figures/k-means.pdf")
image(cl,fastcol=FALSE,graph=TRUE)

image(cl,fastcol=FALSE,graph=TRUE)
points(X.train,col=pred_train,cex=.3)

image(cl,fastcol=FALSE,graph=TRUE)
points(X.train,col=pred_train,cex=.3)
points(X.test,col=pred_test,pch=22,bg="orange")
dev.off()

# Additional K-Means plot with PCA components
scores4<-pca.train$scores
ggdata2<-data.frame(scores4, Cluster=cl@cluster)
png(filename="figures/kmeans_cluster_pca.png", width =
1024, height = 768)
ggplot(ggdata2) + geom_point(aes(x=Comp.1, y=Comp.2,
color=factor(Cluster)), size=5, shape=20) +
stat_ellipse(aes(x=Comp.1,y=Comp.2,fill=factor(Cluster)),
geom = "polygon", level=0.95, alpha=0.4) +
guides(color=guide_legend("Cluster"),
fill=guide_legend("Cluster"))
dev.off()

```