Performance Prediction for Enabling Intelligent Resource Management on Big Data Processing Workflows

Aleksandra Obeso Duque
Abstract

Performance Prediction for Enabling Intelligent Resource Management on Big Data Processing Workflows
Aleksandra Obeso Duque

Mobile cloud computing offers an augmented infrastructure that allows resource-constrained devices to use remote computational resources as an enabler for highly intensive computation, thus improving end users experience. Being able to efficiently manage cloud elasticity represents a big challenge for dynamic resource scaling on-demand. In this sense, the development of intelligent tools that could ease the understanding of the behavior of a highly dynamic system and to detect resource bottlenecks given certain service level constrains represents an interesting case of study. In this project, a comparative study has been carried out for different distributed services taking into account the tools that are available for load generation, benchmarking and sensing of key performance indicators. Based on that, the big data processing framework Hadoop Mapreduce, has been deployed as a virtualized service on top of a distributed environment. Experiments for different cluster setups using different benchmarks have been conducted on this testbed in order to collect traces for both resource usage statistics at the infrastructure level and performance metrics at the platform level. Different machine learning approaches have been applied on the collected traces, thus generating prediction and classification models whose performance is then evaluated and compared. The highly accurate results, namely a Normalized Mean Absolute Error below 10.3% for the regressor and an accuracy score above 99.9% for the classifier, show the feasibility of the prediction models generated for service performance prediction and resource bottleneck detection that could be further used to trigger auto-scaling processes on cloud environments under dynamic loads in order to fulfill service level requirements.
# Table of Contents

List of Tables ........................................ IV

List of Figures ........................................ V

List of Acronyms ........................................ VIII

1 Introduction ........................................ 1
   1.1 Background ................................... 1
   1.2 Problem and Contributions .................. 1
   1.3 Project scope ................................ 3
   1.4 Outline ..................................... 3

2 Theoretical Framework .............................. 5
   2.1 Cloud Computing .............................. 5
      2.1.1 Contextualization and Orchestration ...... 5
      2.1.2 Virtualization ............................ 6
      2.1.3 Autonomic Computing .................... 6
      2.1.4 Cloud Performance and Efficiency ...... 7
   2.2 Big Data and Hadoop .......................... 9
      2.2.1 Hadoop Distributed File System ......... 11
      2.2.2 Yet Another Resource Negotiator ....... 12
      2.2.3 MapReduce .............................. 13
   2.3 Machine Learning ............................... 15
      2.3.1 Regression versus Classification ....... 15
      2.3.2 Supervised versus Unsupervised Learning 16
      2.3.3 Model Generation Workflow ............. 16
      2.3.4 ML Issues ................................ 20
      2.3.5 Methods ................................. 21

3 Related Work ........................................ 25

4 Methodology ......................................... 30
   4.1 Testbed Set-up ................................ 30
      4.1.1 Hardware Resource Identification ....... 30
      4.1.2 Cloud Service Decision .................. 30
      4.1.3 Testbed Design and Deployment .......... 31
   4.2 Experimental Design ........................... 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Data Extraction Pipeline</td>
<td>35</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Data Collection and Parsing</td>
<td>35</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Model’s I/O Definition</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>Tools for Data Analytics</td>
<td>38</td>
</tr>
<tr>
<td>4.5</td>
<td>Generation of I/O Traces</td>
<td>38</td>
</tr>
<tr>
<td>4.5.1</td>
<td>System Activity Report (SAR) Interpreter</td>
<td>38</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Job History (JHist) Interpreter</td>
<td>39</td>
</tr>
<tr>
<td>4.6</td>
<td>Pre-processing</td>
<td>39</td>
</tr>
<tr>
<td>4.6.1</td>
<td>Traces Concatenation</td>
<td>39</td>
</tr>
<tr>
<td>4.6.2</td>
<td>Feature Selection</td>
<td>40</td>
</tr>
<tr>
<td>4.6.3</td>
<td>Over-sampling</td>
<td>40</td>
</tr>
<tr>
<td>4.6.4</td>
<td>Data Split</td>
<td>40</td>
</tr>
<tr>
<td>4.6.5</td>
<td>Feature Scaling</td>
<td>41</td>
</tr>
<tr>
<td>4.6.6</td>
<td>Dimension Reduction</td>
<td>41</td>
</tr>
<tr>
<td>4.7</td>
<td>Batch Learning</td>
<td>41</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Regression</td>
<td>41</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Classification</td>
<td>42</td>
</tr>
<tr>
<td>5</td>
<td>Results</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>System Resource Utilization</td>
<td>44</td>
</tr>
<tr>
<td>5.2</td>
<td>System Performance</td>
<td>45</td>
</tr>
<tr>
<td>5.3</td>
<td>Resource Utilization and Carried Load Traces</td>
<td>47</td>
</tr>
<tr>
<td>5.4</td>
<td>Resource Utilization and Carried Load Spaces</td>
<td>47</td>
</tr>
<tr>
<td>5.5</td>
<td>KPI Data Distribution</td>
<td>52</td>
</tr>
<tr>
<td>5.6</td>
<td>Batch Learning</td>
<td>54</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Regression</td>
<td>54</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Classification</td>
<td>62</td>
</tr>
<tr>
<td>6</td>
<td>Discussion &amp; Analysis</td>
<td>66</td>
</tr>
<tr>
<td>6.1</td>
<td>Model Comparison</td>
<td>66</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Regression</td>
<td>66</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Classification</td>
<td>66</td>
</tr>
<tr>
<td>6.2</td>
<td>Challenges</td>
<td>67</td>
</tr>
<tr>
<td>6.3</td>
<td>Connection with Previous Work</td>
<td>68</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>70</td>
</tr>
<tr>
<td>Literature</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>A</td>
<td>Demonstrator Development</td>
<td>76</td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1: Binary Classification Performance: Confusion Matrix. .......... 19
Table 4.1: Service Comparison .................................................. 32
Table 4.2: SLA definition. ......................................................... 37
Table 4.3: SLA example thresholds. .............................................. 38
Table 4.4: Samples Collected per Experiment: Imbalance in Regression. .......... 40
Table 4.5: Samples per SLA class: Imbalance in Classification. .............. 40
Table 5.1: Performance: MLP Regressor. .................................. 54
Table 5.2: Performance: Lasso Regressor. .................................. 57
Table 5.3: Performance: RF Regressor. ..................................... 60
Table 5.4: Performance: MLP Classifier (Note that there are two values per metric,
one per class). ................................................................. 63
Table 5.5: Performance: Ridge CV Classifier. .............................. 64
Table 5.6: Performance: RF Classifier. ...................................... 64
Table 6.1: Regressors comparison. ............................................ 66
Table 6.2: Classifiers comparison. ............................................ 67
List of Figures

Figure 1.1: Analysis Component for Automated Enforcement of Cloud SLA. ... 3
Figure 2.1: Classical Auto-scaling Workflow. ........................................ 7
Figure 2.2: Cloud SLA. ........................................................................ 9
Figure 2.3: Hadoop System Design. ...................................................... 11
Figure 2.4: HDFS Architecture. .............................................................. 12
Figure 2.5: MapReduce Workflow. ......................................................... 14
Figure 2.6: SGD. ................................................................................ 18
Figure 2.7: ANN: Neuron. ................................................................. 23
Figure 2.8: Recursive Binary Splitting and its Tree Representation. ......... 24
Figure 4.1: System Prototype. ............................................................... 30
Figure 4.2: Service Architecture Design: SM and SW. ......................... 31
Figure 4.3: Job profiles: WC and TS: Map tasks are shown in red (failed tasks are shown in gray), shuffle in orange, sort in yellow and reduce tasks are shown in green. ........................................... 34
Figure 4.4: Data Extraction Pipeline. ..................................................... 35
Figure 4.5: Trace Generation Pipeline. ................................................... 38
Figure 4.6: Pre-processing Pipeline. ...................................................... 39
Figure 4.7: Machine Learning Approaches for Regression and Classification. .... 41
Figure 5.1: % CPU Utilization for 1M3S and 32 GB. ............................... 44
Figure 5.2: % Memory Utilization for 1M3S and 32 GB. ......................... 44
Figure 5.3: % Network Utilization for 1M3S and 32 GB. ....................... 45
Figure 5.4: Disk I/O Utilization (TPS) for 1M3S and 32 GB. .................. 45
Figure 5.5: Input Data Size vs. Elapsed Time for different Cluster Set-ups. ... 46
Figure 5.6: Input Data Size vs. Max. Map Throughput for different Cluster Set-ups. ... 46
Figure 5.7: Shuffle Data Size vs. Max. Shuffle-Sort-Reduce Throughput for different Cluster Set-ups. .............................. 46
Figure 5.8: RU and CL Traces when varying Input Data Size: WC. ......... 48
Figure 5.9: RU and CL Traces when varying Input Data Size: TS .......... 49
Figure 5.10: RU and CL Spaces: WC. ................................................. 50
Figure 5.11: RU and CL Spaces: TS. ..................................................... 51
Figure 5.12: PDF generated using KDE: WC. ...................................... 52
Figure 5.13: PDF generated using KDE: TS. ....................................... 53
Figure 5.14: Expected vs. Predicted CL using MLP Regression: WC. ....... 55

© Uppsala University  Aleksandra Obeso Duque
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.15</td>
<td>Expected vs. Predicted CL using MLP Regression: TS</td>
<td>56</td>
</tr>
<tr>
<td>5.16</td>
<td>Expected vs. Predicted CL using Lasso Regression: WC</td>
<td>58</td>
</tr>
<tr>
<td>5.17</td>
<td>Expected vs. Predicted CL using Lasso Regression: TS</td>
<td>59</td>
</tr>
<tr>
<td>5.18</td>
<td>Expected vs. Predicted CL using RF Regression: WC</td>
<td>61</td>
</tr>
<tr>
<td>5.19</td>
<td>Expected vs. Predicted CL using RF Regression: TS</td>
<td>62</td>
</tr>
<tr>
<td>5.20</td>
<td>Expected vs. Predicted Classification using MLP: c. Note that the red points denote SLA Violation whereas green denote SLA compliance</td>
<td>63</td>
</tr>
<tr>
<td>5.21</td>
<td>Expected vs. Predicted Classification using Ridge CV: SLA</td>
<td>64</td>
</tr>
<tr>
<td>5.22</td>
<td>Expected vs. Predicted Classification using RF: SLA</td>
<td>65</td>
</tr>
</tbody>
</table>
### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M1S</td>
<td>1 Master - 1 Slave</td>
</tr>
<tr>
<td>1M2S</td>
<td>1 Master - 2 Slave</td>
</tr>
<tr>
<td>1M3S</td>
<td>1 Master - 3 Slave</td>
</tr>
<tr>
<td>1M4S</td>
<td>1 Master - 4 Slave</td>
</tr>
<tr>
<td>AC</td>
<td>Autonomic Computing</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ARIA</td>
<td>Automated Resource Inference and Allocation</td>
</tr>
<tr>
<td>AS</td>
<td>Adaptive Scheduler</td>
</tr>
<tr>
<td>AUC</td>
<td>Area Under the Curve</td>
</tr>
<tr>
<td>CAP</td>
<td>Consistency, Availability and Partition Tolerance</td>
</tr>
<tr>
<td>CDF</td>
<td>Cumulative Distribution Functions</td>
</tr>
<tr>
<td>CLS</td>
<td>Carried Load Space</td>
</tr>
<tr>
<td>CL</td>
<td>Carried Load</td>
</tr>
<tr>
<td>CV</td>
<td>Cross-Validation</td>
</tr>
<tr>
<td>DC</td>
<td>Data Center</td>
</tr>
<tr>
<td>EC2</td>
<td>Amazon Elastic Compute Cloud</td>
</tr>
<tr>
<td>EDF</td>
<td>Earliest Deadline First</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-in First-out</td>
</tr>
<tr>
<td>FN</td>
<td>False Negative</td>
</tr>
<tr>
<td>FPR</td>
<td>False Positive Rate</td>
</tr>
<tr>
<td>FP</td>
<td>False Positive</td>
</tr>
<tr>
<td>GW</td>
<td>Gateway</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop Distributed File System</td>
</tr>
<tr>
<td>HTM</td>
<td>Hierarchical Temporal Memory</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>JHist</td>
<td>Job History</td>
</tr>
<tr>
<td>KDE</td>
<td>Kernel Density Estimate</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>LBFGS</td>
<td>Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm</td>
</tr>
<tr>
<td>LB</td>
<td>Load Balancer</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long-Short Term Memory</td>
</tr>
<tr>
<td>LXC</td>
<td>Linux Containers</td>
</tr>
<tr>
<td>MASE</td>
<td>Mean Absolute Square Error</td>
</tr>
</tbody>
</table>
### List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDN</td>
<td>Mixture Density Networks</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-layer Perceptron</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MR</td>
<td>Map-Reduce</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Square Error</td>
</tr>
<tr>
<td>NMAE</td>
<td>Normal Mean Absolute Error</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>NoSQL</td>
<td>Not Only Structured Query Language</td>
</tr>
<tr>
<td>OML</td>
<td>Online Machine Learning</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PDF</td>
<td>Probability Density Function</td>
</tr>
<tr>
<td>PM</td>
<td>Provisioning Manager</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional-Integral-Derivative</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RF</td>
<td>Random Forest</td>
</tr>
<tr>
<td>ROC</td>
<td>Receiver Operating Characteristic</td>
</tr>
<tr>
<td>RTP</td>
<td>Real-time Transport Protocol</td>
</tr>
<tr>
<td>RUS</td>
<td>Resource Utilization Space</td>
</tr>
<tr>
<td>RU</td>
<td>Resource Utilization</td>
</tr>
<tr>
<td>SAR</td>
<td>System Activity Report</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descend</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SLO</td>
<td>Service Level Objective</td>
</tr>
<tr>
<td>SML</td>
<td>Statistical Machine Learning</td>
</tr>
<tr>
<td>SM</td>
<td>Swarm’s Manager</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>SVR</td>
<td>Support Vector Regression</td>
</tr>
<tr>
<td>SW</td>
<td>Swarm’s Worker</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>TN</td>
<td>True Negative</td>
</tr>
<tr>
<td>TPR</td>
<td>True Positive Rate</td>
</tr>
<tr>
<td>TPS</td>
<td>Transactions per Second</td>
</tr>
<tr>
<td>TP</td>
<td>True Positive</td>
</tr>
<tr>
<td>TS</td>
<td>Terasort</td>
</tr>
<tr>
<td>vCPU</td>
<td>Virtual CPU</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VRC</td>
<td>Virtual Resource Controller</td>
</tr>
<tr>
<td>WC</td>
<td>Wordcount</td>
</tr>
<tr>
<td>YARN</td>
<td>Yet Another Resource Negotiator</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Background

In computer systems, predicting the amount of resources to allocate in order to ensure Quality of Service (QoS) is intrinsically a difficult task, especially if it is approached using analytical models that require a thorough understanding of the system’s components and their interactions. Factors such as dynamic user demand, complex applications, virtual machines (VM) co-hosted on general purpose platforms that do not guarantee real-time performance [1], distributed infrastructure, among others, determine the system’s complexity. Another way to approach this problem can be done using Machine Learning (ML) methods, in order to learn the system’s behavior from representative observations. This approach does not require a deep knowledge of the system’s components and their interactions, thus making possible to develop data-driven, intelligent and automated decision systems.

The framework of this project has as main goal setting-up an experimental scenario, collect relevant metrics, data analysis and evaluation for cloud-service performance estimation based on different ML techniques. The objective is to define and test different prediction models for a set of resource types, mapping resource usage to service performance as a way to monitor QoS on a virtualized and distributed test-bed [1]. These prediction models will enable in the future, proactive resource dimensioning at the time of deploying cloud services, that combined with system anomaly detection, resource exhaustion alerts, among others, as the components of an end-to-end automated service assurance system for Service Level Agreement (SLA) compliance.

1.2 Problem and Contributions

Mobile cloud computing integrates cloud elasticity and quality of mobile services in order to offer an augmented infrastructure that allows resource-constrained devices to use remote computational resources as an enabler for highly computational-intensive mobile applications, thus improving end users experience [2]. Being able to efficiently manage cloud capacity represents a big challenge for the dynamic allocation of cloud resources based on a dynamic demand. Real-time prediction models constitute a key capability for leveraging real-time service assurance. In this context, cloud capacity is an important concept to determine the load that can be handled by the available infrastructure without incurring into SLA violations [3].
There exist a big gap on the way clients specify service level requirements and the resource-centric systems offered and monitored by cloud providers. This gap makes necessary to translate resource requirements to service quality expectations in order to avoid an inefficient resource usage and/or incurring into SLA infringements. This problem can be tackled by helping the distributed system to know the load that can be carried for a specific service, thus mapping resource usage statistics to service level performance metrics improving the usability of cloud systems. The features included in cloud systems allow providers to have a high flexibility regarding which and how many resources to dedicate for a specific service, having into account that several resource combinations can yield the same service performance [4].

Resource over-provisioning as a way to cope for feature peak loads does not represent a very efficient approach from an energy consumption perspective, since resources are most of the time underutilized knowing that is possible to assign these idle resources to other services. On the other hand, resource under-provisioning represents a risk for the agreed QoS, where the expected performance of the required resources is not always fulfilled thus inducing SLA violations. The most efficient approach would be to adjust the provisioned resources based on the dynamic demand, thus providing both good performance and cost-efficient resource utilization.

Real-time service assurance will represent a key feature for the next generation mobile services that integrate distributed cloud computing capabilities as a way to offer high scalability, low-delay and autonomic systems as support for applications running on resource-constrained devices, typically common in Internet of Things (IoT) use cases. This problem wraps several challenges that should be taken into account at the time of dealing with cloud-based services. Some of these challenges include the overhead caused by spawning new Virtual Machines (VMs), accurate prediction models for dynamic workloads, computation of the right number of resources needed based on the available flavors 1 [5]. Ideally, to approach these challenges, proactive strategies such as a-priori resource allocation are needed when a peak workload is predicted.

The inherent uncertainties of big data processing, characterized by variable volumes and velocities with dynamic load patterns on an elastic distributed computing platform, pose a very interesting case of study. This is especially interesting when there is a need of guaranteeing the QoS provided while keeping low costs. Thus, understanding the relationship between the resource utilization and QoS metrics, helps towards building up autonomous and highly scalable systems powered by self-aware and self-regulating features. From the Service Provider (SP) perspective, providing a service in compliance with the SLA while having an efficient usage of the resources available in the infrastructure, avoiding under- or over-provisioning represents an important capability.

In this project, a big data processing framework is deployed as an example of a platform service running on top of a virtualized and distributed infrastructure. On this test-bed, different experiments are run in order to stress and collect infrastructure

---

1Flavor as the available resource configuration of a particular VM.
level usage metrics and platform level performance metrics as the observations used to learn different ML-based prediction models. These models can be used as part of a self-awareness module useful for Infrastructure as a Service (IaaS) providers as a way to predict QoS on services running at the platform level. At the same time, this information can be used to efficiently know when to trigger scaling processes to cope with dynamic workloads while meeting the agreed SLA and, at the same time, making an efficient use of the available resources. Fig. 1.1 shows the contribution of this project as an analysis module in an automated system for cloud SLA enforcement.

![Analysis Component for Automated Enforcement of Cloud SLA.](image)

**Figure 1.1**: Analysis Component for Automated Enforcement of Cloud SLA.

### 1.3 Project scope

The scope of the proposed thesis project does not intend to develop an end-to-end automated service assurance system able to make real-time decisions for dynamic resource allocation and service deployment. It aims to continue the improvement and evaluation of the approach presented in [3] in order to predict SLA conformance for cluster-based services using ML, considering different load scenarios and a virtualized service in different distributed settings.

### 1.4 Outline

This document is made up of seven chapters. Chapter 1 contains an introductory section where a short background is given, statement of the problem to deal with, the contributions of this project on that framework and the project delimitations. Chapter 2 contains the theoretical framework related to the problem such as distributed systems, big data, SLA enforcement for cloud-based services and ML related topics. Chapter
3 is an overview of the state of the art related to this project including estimation of resource utilization in single or multiple machines using both analytical and data-driven methodologies. Chapter 4 explains the methodology used to approach the problem, from setting up the test-bed, running experiments, data collection and analysis using different ML prediction models. Chapter 5 presents the results obtained with the different learned models for both regression and classification problems. In chapter 6 the results presented are discussed comparing the performance achieved with different models in order to see what is the most appropriate ML-based approach to use for this scenarios. Chapter 7 states the conclusions after analyzing the results of this project and presents the future work that is foreseen in order to enhance the results of this approach and the next steps to take towards an automated system.
2 Theoretical Framework

2.1 Cloud Computing

Cloud Computing refers to a pool of configurable computing resources that can be rapidly provisioned and released. These resources are exposed to the user via the network as an abstract representation of data storage, servers, networks, applications, etc. Usually these resources are exposed as services via Application Programming Interfaces (APIs), in a transparent way, abstracting sub-layered details in a way that users do not have to worry about underlying details. Clients use these resources using a pay-as-you-go model where services are automatically spawned on-demand without the intervention of the service provider. In this way, cloud computing is well known as a good approach to offer highly scalable services; if more resources are needed at a particular time, the required services are deployed on the fly and, in the opposite scenario, they are freed in case they are no longer needed, making them available for other purposes [6].

Cloud computing is usually represented by three main levels: IaaS, Platform as a Service (PaaS) and Software as a Service (SaaS) [7]. IaaS provides fundamental computing resources such as computation, storage, network. PaaS provides tools such as programming languages, libraries, frameworks, development environments, as a support for the deployment of customized applications. SaaS offers applications as services running on top of the cloud infrastructure. There are also different deployment models: public, community, private and hybrid. Public cloud computing allows new intelligent business to grow faster, avoiding the initial costs of acquiring expensive resources to support highly intensive computational systems. Nowadays, there is a big interest in aggregating resources from different deployment models in order to ease the start-up process thus lowering the upfront investment [8].

The main advantage cloud computing brings is related to the fact that it does not require specialized or static worker nodes; services are loosely coupled with the underlying resources by using virtualization and services are available on demand without the need of handling job queues.

2.1.1 Contextualization and Orchestration

Contextualization allows resources to be rapidly provisioned and released with minimal effort and without interacting with the service provider. The rapid deployment of resources is often carried out based virtualization. Customized computing environments
are offered as VMs capable of configuring themselves during the spawning process based on the user set-up; this allows to avoid the storage of fat images [9].

Orchestration enables the automatic contextualization process in order to provide agile deployment, rapid elasticity, easy management and high availability.

### 2.1.2 Virtualization

Virtualization abstracts a single machine’s resources by splitting them into different execution environments that can be run concurrently giving the illusion of having several isolated VMs. The host machine (the physical machine) and the guest machine (the VM) are isolated from each other [10].

VMs allow to pack full software stacks as virtual images that can easily be distributed and migrated on-the-fly between different hosting environments, thus allowing a loose coupling between the software and hardware. These virtual images ease the provision and release processes bringing more interoperability since they allow to have different guest and host operating systems [7]. In conclusion, VMs provide business agility, operational flexibility, high availability and fault-tolerance [10]. Then, the focus can be kept on functionality and still reach different users under different distributions.

Containers are a type of virtualization at the Operating System (OS) level. It represents a light-weight virtualization option but provides lower isolation when compared to VMs. With this type of virtualization, the kernel space is shared and the user space is separated from each other [11]. This type of virtualization is very appropriate when dealing with micro-service architectures. Applications that are deployed as a set of distributed micro-services on interconnected containers, ease the replication process as a way to provide highly available systems, fast debugging and high scalability [10].

Unlike VMs, containers do not provide full isolation among the coexisting environments, thus is not a very suitable when security is a concern. However, they are characterized by a low overhead while deploying new instances. Examples of container platforms are Linux Containers (LXC) and Docker (a set of tools that provides automated deployment of applications on containers).

### 2.1.3 Autonomic Computing

Cloud auto-scaling or elasticity is a dynamic provisioning mechanism that allows to allocate or release the resources on the fly in order to meet the run-time requirements posed by dynamic workloads. A classical auto-scaling workflow is depicted in Fig. 2.1. In Amazon, for example, the developer can define auto-scaling groups of resources that can be scaled-out when an event is triggered by an increased load or scaled-in when the load is reduced. Tools such as Amazon CloudWatch can be used to monitor the resource utilization and policies given by the developer configuring the thresholds in charge of triggering the scaling events and the actions to be taken can be easily specified.
Autonomic Computing (AC) makes reference to the development of computer systems capable of managing resources automatically in highly complex and distributed scenarios. The main idea is to self-manage computer system in a transparent way thus freeing system administrators of carrying out low level management task. These systems also include the ability to adapt upon dynamic changes, triggered by load or configurations. An AC system usually includes sensors that are monitoring the status of the system under a dynamic load, effectors that are in charge of self-adjusting the system to cope with changes and a controller that could be implemented using ML or more classical approaches such as Proportional-Integral-Derivative (PID) controllers. All of these components integrate an expert system that applies certain policies based on self- and environment awareness.

Distributed systems and the requirements of distributed applications are highly complex and their management is very hard to control by system administrators since it is time consuming and even error-prone. In AC, system administrators have another goal, they are in charge of configuring general or high level policies in order to guide the system during self-management tasks execution. Some of the self-management tasks include self-configuration, self-healing, self-regulation, self-awareness, self-optimization, self-protection, among others, without external intervention [12].

2.1.4 Cloud Performance and Efficiency

Key Performance Indicators

There are different sources of performance variability. Some of these sources are, for example, the hypervisor policy, dynamic load, multi-user environments, data locality, heterogeneous infrastructures in federated clouds, among others. Key Performance Indicators (KPIs) are intended to be used as performance metrics that monitor the service functionality. Based on this indicators, it is possible to define Service Level Objectives (SLOs) or thresholds between the consumer and the SP as part of the SLA. These SLOs can be related to the service performance, availability or fault tolerance [13]. KPI are important indicators of the QoS [14]. Examples of these indicators are [15]:

© Uppsala University Aleksandra Obeso Duque
• Performance measured as response time or throughput that gives information regarding the cloud service performance.

• Availability as the property of a service able to be accessible and usable upon demand. It is usually measured in terms of uptime, percentage of successful requests, percentage of requests, etc.

• Reliability measured in terms of the mean time between failures or the mean time to recover from failures.

• Cloud capacity measures the maximum load that can be held for an individual customer, in terms of the available core resources and does not necessary reflect the overall supported capacity of the service provider. In this sense, cloud service customers should be able to dynamically decide how much capacity to get in order to support their service load. There are different metrics related to cloud capacity, such as number of simultaneous connections, maximum resource capacity, service throughput, etc [15]. In this study, processing capacity is a key concept expressed as the amount of information that can be processed per unit time [16].

• Scalability is the ability of the system to efficiently manage a dynamic amount of workload [17]. All resources can be scaled in such a way that they are highly elastic, expanding or shrinking them based on a dynamic demand. Vertical scaling means scaling up/down by increasing or reducing the capacity at each of the available nodes. On the other hand, horizontal scaling refers to scaling in/out resources, which is more common in cloud computing where the main objective is to add more working nodes to allow features such as data replication and distributed computing.

There has been a huge interest in auto-scalable systems with the expectation to have future systems powered by ML in order to efficiently manage resources under highly dynamic scenarios. This offers the possibility to have scalable distributed applications featured with rapid elasticity as the capability of provisioning and releasing services automatically according to a dynamic demand. In this sense, resources seem to be unlimited for the user of the service.

This KPI can be measured based on strong and weak scaling capabilities of the system. Strong scaling is measured comparing the execution time as more resources are added given a fixed amount of work. Weak scaling tries to keep the amount of work per processor constant, studying how the execution time behaves when adding more resources and proportionally increasing the work load [18].

Service Level Agreement

SLA is a policy between the service provider and the client of a particular service (whether IaaS, PaaS, SaaS) stating the QoS as a commitment from the SP to the client.
and it also states information about penalization in case of violation. A SLA is usually expressed in terms of qualitative or quantitative metrics known as SLOs. SLOs express boundaries on the behavior of a cloud service and its limitations. These SLOs can be used during runtime in order to monitor the system functionality, thus allowing to take corrective measures in real time [15].

When comparing to other types of services, cloud-based services have to define SLAs that consider the highly dynamic environment of the system and the need of automated mechanisms to negotiate these SLAs with the customer without the need of interacting directly with the SP, for example by using Smart Contracts.

The clients for SaaS are end-users of the software applications on top of the cloud. Clients for PaaS can be software developers, application designers. Clients for IaaS can be system designers or architects [13].

Different SLA metrics have been defined for each cloud level. For IaaS, SLA metrics can be expressed as CPU, storage, memory capacity, availability, response, time, scaling capabilities. For PaaS, it can be expressed as scalability, ease of integration [13], etc. For SaaS, it can be expressed in terms of availability, reliability, scalability. In this way, the SLA is orthogonal to the cloud models and can be applied to any of the services offered in each abstraction layer, whether IaaS, PaaS or SaaS [14], as depicted in Fig. 2.2.

![Figure 2.2: Cloud SLA.](image)

In the typical scenario, the SLA is given in terms of the service availability but there is no commitment regarding response time thus performing based on best effort.

### 2.2 Big Data and Hadoop

Nowadays, the data explosion is a fact, due to end-devices collecting a huge amount of data from sensors at very high rates. Data is expanding on the following three fronts:
velocity, variety and volume. In terms of volume, even though there are storage disks capable of storing large datasets, the I/O rate usually cannot cope with the current requirements when handling applications in real time for data stream processing [19].

Parallel and distributed computational platforms have been used in order to process this huge amount of data in a faster way, these platforms are usually highly scalable and highly resilient as a way to recover from faults. These platforms come as a support to generating valuable information in real-time so that important decisions can be made on-time.

The IoT era has brought important challenges with the explosion of data generated and collected from different sensors supporting smart cities. This information needs to be processed in real or near real time in order to generate valuable knowledge.

In terms of variety, data is collected from different applications and in several formats, so that it requires to apply pre-processing pipelines where data is filtered, converted, or in general, transformed to become more structured and inter-operable. The data collected can be structured, semistructured or unstructured.

When handling big data, important considerations shall be taken into account such as the heterogeneity of data that is either collected or generated. The velocity at which the data is streamed shall also be considered.

Usually, big data is a very general concept that applies well when the data that is being handled cannot be stored, processed using commodity systems or traditional data base solutions.

In order to support the digitalization era, there is a need of developing horizontal scalable systems from which big data applications can be developed thus allowing real time processing. Often, to cope with big data applications, the data is stored in a distributed way. This can become a problem since data consistency can be seen as a limitation.

Not Only Structured Query Language (NoSQL) comprises next generation database systems capable to handle non-relational, distributed and horizontal scalable database systems. It is based on the Consistency, Availability and Partition Tolerance (CAP) theorem, where only two of these properties can be achieved at the same time, compromising one of them according to the specified priority for a particular application. Usually, this usually provides eventual consistency [20].

Big data analytics can help telecommunication service providers to improve processing efficiency for a better user experience. Insights or patterns can also be deduced or predicted in order to make decisions after analyzing big volumes of data that travel on their network. It can improve profitability by enhancing user experience, optimizing network usage by predicting periods of high demand, predicting failures or low performance.

Hadoop is an open source implementation of the MapReduce framework originally proposed by Google for distributed processing and storage of big data [21]. It provides high scalability, fault-tolerance and simplicity. Among the different components, Hadoop is made up of a distributed file system named Hadoop Distributed File System (HDFS),
the MapReduce framework, the resource management platform Yet Another Resource Negotiator (YARN) and some common utilities, all of them originally designed for distributed processing on homogeneous environments [16] (Fig. 2.3).

![Diagram of Hadoop System Design](https://en.wikipedia.org/wiki/Apache_Hadoop)

Figure 2.3: Hadoop System Design (https://en.wikipedia.org/wiki/Apache_Hadoop).

An Application in Hadoop makes reference to a set of tasks that are run together. A collection of physical resources is called Resource Container. These containers can be allocated for the execution of applications and be released once they are done. YARN is not only used on MapReduce tasks but also for tasks from other frameworks such as Spark.

In order to avoid networking bottlenecks caused by transferring large amounts of data, recent approaches move the computation where data is located. There are different types of data locality, i.e:

- **Locally**: When data is placed locally where the computation is generated.
- **Intra-rack**: When data is located in the same rack but in another node different to the one where the computation is generated.
- **Inter-rack**: When data is located in a different rack from the node’s rack where the computation is generated.

### 2.2.1 Hadoop Distributed File System

Usually, distributed file systems are made up of aggregated resources using an abstracted way that allows to think of it as a single global file system. It usually supports data replication as a way to provide fault-tolerance, concurrent access management, among others [20].

HDFS designed for Apache (Fig. 2.4), is a highly scalable and fault-tolerant system very suitable for providing big data storage that supports data streaming on commodity
hardware. It has been designed for batch processing, allowing high aggregated throughput instead of providing low latency. It supports single write and concurrent read of files.

Data is split into smaller units or blocks that are distributed and replicated on different nodes, thus allowing high availability and fault-tolerance.

Its distributed architecture comprises a namenode at the master node, an optional secondary namenode and several datanodes (slave nodes). The namenode is in charge of handling metadata and maintaining the file system namespace. Datanodes are in charge of handling the file blocks and their replication, and also allow read-write operations.

![Figure 2.4: HDFS Architecture.](image)

### 2.2.2 Yet Another Resource Negotiator

YARN is in charge of assigning computational resources for application execution on Hadoop. It is made up of three main components:

- A Resource Manager on the master node in charge of monitoring available resources, running the scheduler and other important services.

- An Application Master per running application that is in charge of monitoring and communicating the application execution state to the resource manager. It also negotiates containers with the Resource Manager in case they are needed.

- Node Managers on the slave nodes are in charge of monitoring containers stages thus monitoring task execution on MapReduce workflows.

YARN handles different schedule policies such as [23]:

- First-in First-out (FIFO) scheduler: Jobs are attended assigning a higher priority for the ones that are submitted earlier.

- Fair scheduler: Assigns at least a predefined number of slots to each job once they are submitted.
• Capacity scheduler: Allows to work with very large clusters ensuring the demanded capacity for each user. Resources are grouped in pools and users are assigned to each pool, handling a FIFO strategy but considering priorities per queue [24].

2.2.3 MapReduce

[25] Hadoop MapReduce is a framework that allows specifying and processing large datasets in a parallel and distributed way with a high abstraction level, thus reducing the overall execution time when compared to sequential processing on a single node. The programmer does not have to worry about complex low-level details such as handling concurrency, fault-tolerance, etc. The processing is brought to where the data is stored, which represents a very suitable option when dealing with big data processing. The benefit of processing the data where it is originally located is the reduction of the network overhead thus providing higher scalability. From this perspective, the framework has been designed to provide high computational throughput but limited network throughput. MapReduce is usually offered as PaaS [14] and it is recommended to use only for long-running jobs where the overhead caused by startup time, disk I/O, communication bottleneck, that are intrinsically caused by the framework operation, can be neglected.

The main stages when a user submits a MapReduce job are:

• Split: Partitions the input data into smaller chunks to be stored along several cluster nodes to be processed in a distributed and parallel way by the map stage.

• Map: Maps an input dataset into a collection of pairs. Specifies what parallel processing to apply on each data chunk and emits a list of intermediate results. Usually, processing such as filtering or transformations are applied at this stage.

• Shuffle: It is in charge of sorting values based on keys and sending these as the reduce stage input.

• Reduce: At the reduce stage, the partial results are parallely combined based on their keys. This stage cannot start before the map stage is fully done.
Fig. 2.5 shows the MapReduce workflow. The top-level unit of work is called job and it is made up of one or more map/reduce tasks. The master node assigns mapper and reducer tasks to the workers. The mapper reads the assigned file chunk, applies the map function, writes partial results on local disk and informs the master about the result location. Reducers pull the results from mapper disk location and once the mapper stage is done, the data is sorted based on keys. The reduce function is then applied to each key and the final result is appended to the final file located in the distributed file system. The master node is in charge of keeping track of the map/reduce tasks progress, keeps track of the results regarding location and size, detects failures and re-runs tasks when needed.

This abstract uses key-value pairs that can be defined based on any desired data type, thus allowing high flexibility. The user is in charge of implementing both the map and reduce interfaces according to the customized processing to be applied on the provided data.

The shuffle/sort stage does not start before the first map task has been completed. The shuffle stage can be finished only when all the map tasks associated with a particular key are finished and sorted. Shuffle includes networking latencies when data is transferred between nodes. The reduce stage only starts when the shuffle stage is done [23].

The job completion can be sped-up by allocating more resources for both map and reduce stages.

When deploying the MapReduce framework, it is preferred to use a large number of low specification nodes instead of a few with high specifications.

In order to ease the detection of bottlenecks, the MapReduce framework exposes several counters during the job execution to monitor how many times a specific event has occurred. Hadoop MapReduce exposes different levels of counters including job, task and custom counters [26].

Job counters include the time spent in all map and reduce tasks as a performance profile. It also gives information about the number of failed map and reduce tasks per
job as an error metric. An entire job fails if the maximum amount of map/reduce tasks is exceeded based on the MapReduce configuration. Another job counter specifies the number of aggregated map execution based on the data locality.

Task counters track aggregated task execution including the number of input records for reduce tasks, the number of records that are spilled to disk as a saturation metric and the time spent on garbage collection.

Custom counters are a way for the user to implement customized counters for a given application thus allowing the user to specify finer or courser counters.

2.3 Machine Learning

ML is the mechanism by which machines learn from observations or data collected from experimentation considering the uncertainties on that data. It comprises a set of methods, sometimes inspired by how biological systems learn from their interaction with other systems in the surrounding environment. It belongs to the intersection of computer science, statistics, data science, among others. Through inference, relations between a set of representative input features $X$ and a set of output targets $Y$ is learned, thus generating a prediction model. These models express the dependencies between input and outputs.

2.3.1 Regression versus Classification

The regression problem corresponds to the inference of models (function approximation) where the output targets are of quantitative or continuous type [27, App. D], whereas in a classification problem, the targets are of qualitative type mapping any input $X$ to a specific class $\hat{f}(X) \in \{1, ..., K\}$. These classes $K$ separated by decision boundaries or discriminants are typically encoded, for example using binary values [27, Ch. 4] [28, Pag. 28].

The statistical model for regression can be expressed in terms of the real model and the inference error $\epsilon$ as in 2.1, whereas a classification model can be expressed also considering the classes $k$ as in 2.2

$$Y = \hat{f}(X) + \epsilon$$

$$Pr(Y = k | X = x), \quad k = 1, ..., K$$

For binary classification, $\hat{f}(X) \in \{0, 1\}$:

$$\hat{f}(X) = \begin{cases} 
1 & \text{if } P(Y = 1 | X) > r, \\
0 & \text{otherwise}
\end{cases}$$

(2.3)

where the threshold $r$, $0 \leq r \leq 1$, can be specified.
There are different types of both regression and classification models, ranging from very simple linear regression to more advanced ones such as using deep learning; some of them will be introduced in the next subsections.

2.3.2 Supervised versus Unsupervised Learning

[28, Pag. 26]

In supervised learning, both input and outputs are used in order to learn the model, for both regression and classification problems. On the other hand, in unsupervised learning the model itself tries to find patterns or structural properties during the learning process from unlabeled data, using only the inputs. Unsupervised learning is very useful technique for dimension reduction, clustering, etc.

2.3.3 Model Generation Workflow

Usually, the development of a new model using ML is made up of several stages. According to data science, the data collected from observations during experiments should be pre-processed in such a way that certain issues that will be introduced later can be avoided.

Feature Selection

Feature selection represents a way to reduce the dimensionality of input data. As part of the preprocessing stage, there is a need for choosing the most relevant features from all the metrics collected during experimentation. There are different methodologies that can be used in order to select the most relevant features such as domain knowledge, filtering out redundant or irrelevant features based on for instance variance, etc [27, Ch. 3].

Feature Scaling

Feature scaling or normalization is used in order to express features using similar or comparable ranges. One of the most common strategies used are min-max scaling where each feature value $x$ is replaced using $\frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$, where the minimum and maximum values are calculated for each dimension. Another strategy is called standardization where $x$ is replaced by $\frac{x - m_x}{s_x}$, where $m$ represents the mean and $s$ the standard deviation of $x$ [27, Ch. 3]. Depending on the normalization function used, it can introduce bias thus affecting the model performance.

Dimension Reduction

In order to avoid the curse of dimensionality, several strategies for dimension reduction are used, e.g. analyze the correlation among the features in order to dismiss repeated
Theoretical Framework

patterns or re-projecting to a space of lower dimensionality trying to keep as much information as possible in the data. By reducing the number of features the time and memory requirements posed to process data are less demanding. It also represents a way to visualize the data, eliminating irrelevant features and noise reduction [27, App. B] [28, Pag. 228].

One of the most common strategies used for dimension reduction is called Principal Component Analysis (PCA) whose main goal is to find a number of orthogonal vectors in the input space towards the direction of the greatest variance and project the features to the space represented by those vectors. The first principal component gives information of the direction of the largest variance. The remaining principal components are in the direction of the remaining largest variance so that they are orthogonal to the previous principal components.

Training

The training phase is the stage at which the model is inferred through a learning process fitting the model based on the input and output data. During this process, some kind of optimization metric is used, either minimizing a loss function or maximizing the likelihood function:

- Loss minimization

- Maximum likelihood in which the idea is to maximize a Probability Density Function (PDF) in order to estimate the best parameters of the statistical model based on the observed data.

It is important to know that, from a batch learning approach, the data used for training should not be used during the validation phase in order to be able to detect model over-fitting.

The main goal during the training stage is to minimize the error. One of the well-known methods used to approach the error minimization problem is called Gradient Descent and can be expressed in terms of the training data $x_i, y_i \forall i = 1^n$ the loss function $L(x_i, y_i, \theta)$ and the cost function $J(\theta)$ as:

$$\hat{\theta} = \arg \min_{\theta} J(\theta), \quad J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, \theta)$$

(2.4)

Fig. 2.6 shows the best possible solution $\hat{\theta}^A$ or global minimizer, which is harder to find when compared to any minimal solution, known as a local minimizer. Very often, the inferred models get stuck in a local minimizer.
The set of parameters $\theta$ to adjust during the training process defines the search space during the optimization process while searching for the minimum loss model and it is given by:

$$\theta = [\beta_0, \beta_1, ..., \beta_i]^T$$

(2.5)

The general gradient descent algorithm proceeds as follows:

**Algorithm 1: Gradient Descent**

- **Input:** Learning rate $\gamma$
- **Output:** Model parameters $\theta$

$\theta \leftarrow \theta_0$

repeat

$\theta_{t+1} = \theta_t - \gamma g_t, \quad g_t = \nabla_{\theta} J(\theta)$;  
$t \leftarrow t + 1$

until convergence;

The parameter $\gamma$ or *learning rate* is adjusted in such a way that if the error gets worse or if its value oscillates then $\gamma$ should be reduced and if the error slowly gets better then this value should be increased. While the optimization process tries to reach the minimum, the gradient $\nabla J$ has to be calculated which could lead to high computational demand, specially for Artificial Neural Networks (ANN) made up of several hidden layers, large datasets where the number of samples $n$ is very larger. In order to improve the efficiency of these calculations, the chain rule is used in the so called *Back-propagation Algorithm*. The Stochastic Gradient Descend (SGD) also introduces an efficiency improvement regarding the number of samples used for computing the gradient, thus using a smaller dataset at each iteration. From these, three main approaches during the training or learning process can be used:

- **Batch learning:** Usually, small batches of data are used (mini-batches) at each step. I.e. for ANN, the parameter update is accumulated until all batch data has been presented once, then those parameters are actually updated.

- **Online learning:** When a small batch of data or one data sample is used to train the model *on-the-fly* as new data arrives; e.g. for ANN the parameters are updated after each data point or small batch is presented. This strategy is very useful to adapt the model upon variations.
Validation

[28, Ch. 5] There are different approaches to validate the ML models:

- **Validation Set Approach**: With this approach the data is randomly split into two sets, training and test sets. The training set is used during the learning stage and the validation test is used in order to evaluate the performance of the model.

- **k-fold Cross-Validation**: The dataset is randomly split into k subsets, each of them containing almost the same amount of data. The model is trained k using k − 1 subsets for training, an each time a different subset is used for validation.

- **Leave-One-Out Cross-Validation**: This validation method is a special case based on the k-fold cross-validation method, where k = n, with n being the total number of samples.

Performance Evaluation

[29, Part II, 3] [28, Sec. 2.2] The performance of ML models during and after training can be evaluated in terms of accuracy. The accuracy expresses how close the predicted value $\hat{f}(X)$ is from the true value $Y$. The training error is an indication of the average loss over the training samples, whereas the test error is related to the average loss of a learn model over test samples.

Typical performance metrics for a regression problem can be given as:

- Squared loss: $L(Y, \hat{f}(X)) = (Y - \hat{f}(X))^2$

- Absolute loss: $L(Y, \hat{f}(X)) = |Y - \hat{f}(X)|$

More specifically, the Mean Square Error (MSE) is defined as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

(2.6)

For classifiers, the accuracy can be measured in terms of a misclassification indicator $I$: $L(Y, \hat{f}(X)) = I(Y \neq \hat{f}(X))$. The most accurate classifier is the one capable of inferring the most likely class for a set of inputs. Another kind of metric, known as the confusion matrix, is a matching measure between the predicted and the true class labels. A confusion matrix can be seen in Table 2.1. For binary classification, the confusion matrix is expressed in terms of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) predictions with respect to the true values.

Table 2.1: Binary Classification Performance: Confusion Matrix.

<table>
<thead>
<tr>
<th></th>
<th>$\hat{Y} = 0$</th>
<th>$\hat{Y} = 1$</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y = 0$</td>
<td>TN</td>
<td>FP</td>
<td>N</td>
</tr>
<tr>
<td>$Y = 1$</td>
<td>FN</td>
<td>TP</td>
<td>P</td>
</tr>
<tr>
<td>Total</td>
<td>$\hat{N}$</td>
<td>$\hat{P}$</td>
<td></td>
</tr>
</tbody>
</table>
Some important metrics derived from the confusion matrix are, e.g:

- **True Positive Rate (TPR):** \( \frac{TP}{FN+TP} \)
- **False Positive Rate (FPR):** \( \frac{FP}{FP+TN} \)
- **Precision:** \( \frac{TP}{TP+FP} \)
- **Accuracy:** \( \frac{TP+TN}{TP+TN+FP+FN} \)
- **Recall:** \( \frac{TP}{FN+TP} \)

There are also other metrics such as Receiver Operating Characteristic (ROC) which plots the relation between the TPR and FPR as \( r \) goes from 0 to 1. The Area Under the Curve (AUC) of the ROC plot also represents a compact performance measure for a classifier, for example an AUC = 0.1 corresponds to a random guess.

### 2.3.4 ML Issues

#### Curse of dimensionality

[28, Pag. 238] When the number of dimensions increases, data becomes very sparse in the feature space. Then, trying to find patterns on the data based on density and distance or finding data outliers becomes more difficult. In order to avoid this issue, dimensionality reduction techniques are used.

#### Class Imbalance

Class imbalance refers to the case when the label distribution is non-uniform, thus producing misleading accuracy. There are several ways to handle class imbalance, such as:

- **Over- or under-sampling** as a way to generate a new dataset with even class distribution by duplicating samples from the less represented class in order to have the same number of samples as in the most represented class or removing samples from the most represented one or both at the same time.

- **Cost matrix definition** similar to the confusion matrix assigning a misclassification cost for each case. Performance metrics are redefined in order to make them cost-sensitive.

#### Over-fitting

An over-fitted model is one that has adapted too closely to the training data to the point where it performs poorly with new data. Predictions are memorized and the model lacks generalization capabilities. If a model has too many parameters to be adjusted or it is trained for too long or too little data, the risk of over-fitting is higher.
In order to avoid over-fitting several techniques have been proposed including early stopping, validation set, regularization, and more. One of the most common is probably early stopping which uses cross-validation. The dataset is split in three subgroups: a training set for training, a validation set to decide when to stop and a test set to evaluate the generalization of the final mode. For tree-based methods, strategies such as pre- or post-pruning are commonly used.

When choosing ML models, it is important to consider the generalization capabilities of the models, which is evaluated when testing their performance on unseen data.

Bias versus Variance Trade-off

[28, Pag. 33] It is important to consider the bias vs. variance trade-off, i.e. the risk of over- or under-fitting for both parametric and non-parametric models. When a model is very flexible it has low bias and high variance, which could mean higher risk of over-fitting. The squared bias estimates how close the prediction model is from the true model on average. Whereas the variance corresponds to the expected square deviation of the prediction model from its mean.

Some validation approaches such as cross-validation serve for regularization purposes, thus reducing the risk of over-fitting. When a model is very flexible the training error is smaller but the test error is larger as an indication of over-fitting, thus the model presents poor generalization capabilities when its performance is tested using new data.

2.3.5 Methods

Linear Approach

[28, Ch. 3] This approach tries to generate either regression or classification using a linear model, where the output $Y$ corresponds to a linear combination of the inputs $X_1, X_2 ... X_n$:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$  \hspace{1cm} (2.7)

The parameters $\beta_i$ are learned during the training process, referred as $\hat{\beta}_i$, and then data can be predicted using the inferred model:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + ... + \hat{\beta}_n X_n$$  \hspace{1cm} (2.8)

A common linear approach is known as least squares method.

For classification problems logistic function is used expressed as $f(y) = \frac{e^{y}}{1+e^{y}}$. A linear classifier uses a linear decision boundary for separating the classes as much as possible.

Ridge

[28, Pag. 215] In order to avoid over-fitting by controlling the flexibility of the inferred model, the parameters $\beta$ should be kept as small as possible. In this way, a regularization
parameter $\gamma$ is used in order to penalize large values of $\beta$, this can be seen in Eq. 2.9. This methodology can be used not only with linear regression.

$$\hat{\beta} = \arg\min_{\beta} \|X\beta - y\|_2^2 + \gamma \|\beta\|_2^2$$  \hspace{1cm} (2.9)

The parameter $\gamma$ is found using cross-validation as a regularization method.

**Lasso**

[28, Pag. 219]

Lasso uses a similar idea to Ridge but looks at the 1-norm instead of the 2-norm squared as Eq. 2.10, which can result on sparse solutions.

$$\hat{\beta} = \arg\min_{\beta} \|X\beta - y\|_2^2 + \gamma \|\beta\|_1$$  \hspace{1cm} (2.10)

**Artificial Neural Networks**

[29, Part II, 2] A ANN is a nonlinear model where the predicted output $Y$ is expressed in terms of the inputs $X$ and a set of parameters $\theta$. An ANN is well known as a black-box, universal approximation model. It is made up of simpler and smaller units known as neurons or nodes that can operate in parallel. The information is stored in the interconnections, not in the nodes itself. This information conforms the parameters to be tuned in order to infer a particular model and it is represented as weights and biases.

$$S = f(X; \theta)$$  \hspace{1cm} (2.11)

It is based on linear regression, more specifically:

$$S = \beta_0 + \sum_{i=1}^{n} \beta_i x_i = \beta^T X,$$  \hspace{1cm} (2.12)

where $\beta$ is composed by $\beta_i$ the so called weights and $\beta_0$ the bias or intercept, defining a hyperplane.

Introducing a more generalized approach using an activation function $\sigma$, as shown in 2.13. One of most common activation functions are the sigmoid 2.14 and the ReLu 2.15. For classification, a combination of logistic functions 2.16 can be used instead. For a multi-layer perceptron, the activation function on the hidden layers must be non-linear and continuous or differentiable. On the output layer, sigmoid function can be used for classification and linear for regression [29, Part II, 3].

$$Y = f(S) = \sigma(\beta^T X)$$  \hspace{1cm} (2.13)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$  \hspace{1cm} (2.14)
\[ \sigma(x) = \max(0, x) \]  
\[ \sigma(x) = \frac{e^x}{1 + e^x} \]  

Figure 2.7: ANN: Neuron.

A neural network is a combination of several linear regression models as seen in Fig. 2.7 (represented as neurons). The model can be made up of several hidden layers and each layer has several neurons. The prediction model learned by a neural network is better with a deeper network (several layers with few neurons) instead of a shallow and wide (few layers and many neurons).

During the training process, each weight \( b_{ji} \) from node (or input) \( i \) to node \( j \) can be adjusted using the back-propagation algorithm or generalized delta rule as shown in Eq. 2.17, where for an output node \( \delta_j \) is given by Eq. 2.18 and for a hidden node is given by Eq. 2.19. \( d \) represents the desired output and \( y \) the actual output.

\[ \Delta b_{ji} = \gamma \delta_j x_i \]  
\[ \delta_j = f'(S_j)(d_j - y_j) \]  
\[ \delta_j = f'(S_j) \sum_k \beta_{kj} \delta_k \]  

As mentioned before, the learning rate or step length \( \gamma \) can overshoot the minimum or cause oscillations if a large value is used or can get stuck easier in local minimum when a small value is used. Then a momentum term \( \alpha \) 2.20 can be used in the parameter updates in order to smooth the parameter changes over time.

\[ \Delta b_{ji}(t + 1) = \gamma \delta_j x_i + \alpha \Delta b_{ji}(t) \]  

Random Forest

[28, Pag. 319] The tree-based methods are a way to approach the prediction problem from a non-parametric perspective, used for both regression or classification problems, where the input space is split into disjoint regions and for each of them a simple model is trained. For the regression problem, the mean of the training data within that region is used and for the classification problem, the majority vote within the region is used.
An optimal partition is a highly computational intensive problem, then the most efficient way to solve it is by using a greedy approach called *recursive binary splitting* as seen in Fig. 2.8 that splits the data into two subsets based on a particular feature test or condition until some stopping criterion is met, such as the minimum node size or node impurity threshold. Some measures of node impurity or efficiency are used, such as the GINI index, entropy or error-based accuracy.

![Recursive Binary Splitting and its Tree Representation.](image)

The flexibility of a tree-based model is defined by its depth, thus the deeper the tree, the smaller the bias and larger the variance. Then, in order to reduce variance, techniques such as pruning are used to decrease the depth of the tree or to use ensemble methods in order to get an average model from multiple trees. Even though the complexity of ensemble methods increases with the number of trees, it makes it less likely to over-fit.

There are different tree-based models. More specifically, random forest is an ensemble method that builds up several models based on *bagging* or bootstrap aggregating. A general prediction for unseen data is made by aggregating single predictions from the different models. In random forest each split in each tree, a random subset of \( q \leq p \) inputs are considered as splitting variables, thus trying to find the set of inputs and split point that minimizes the loss. Usually, \( q = \sqrt{p} \) is used for classification and \( q = \frac{p}{5} \) for regression. The random input selection used in this method reduces the correlation among trees, thus resulting in an overall loss reduction.
3 Related Work

Some relevant work has already been done in the area of this project. In particular, Ericsson Research Group has developed prediction models of client-side service metrics on single-node [30] and cluster-based [31] topologies for streaming services, based on observations of service metrics and device statistics. Real-time generation of regression models is enabled by the application of on-line statistical machine learning methods on live statistics.

In [30], research was started in order to predict client-side service level metrics based on server-side device statistics (more specifically Linux kernel stats such as CPU, memory, swap and network utilization); in this sense, the generated models are service agnostic. The laboratory testbed is made up of a single server running a video streaming service (VLC), a client side instrumented based on VLC stats including video frame rate, audio frame rate and Real-time Transport Protocol (RTP) packet rate plus a load generator emulating different load patterns (constant, linearly increasing, Poisson, periodic and flash-crowd patterns). The scenario was used to collect several data traces from which dimension reduction techniques were applied to build prediction models using statistical machine learning such as linear regression, lasso regression, regression trees and random forest. To validate the generated models, a validation set was used showing a Mean Absolute Square Error (MASE) at most 15% using random forest trees outperforming the other techniques.

In [31], the laboratory testbed was improved in order to manage a distributed server-side, including networked storage machines and a Load Balancer (LB). In this study, further experimentation was done focusing on feature set reduction using forward-stepwise-selection (reducing the input space from around 840 to 10 features per node). This dimension reduction showed a performance improvement in both accuracy and computational time. An incremental approach was used starting from batch learning, then on-line learning and then it was applied on live stats as a real-time engine introducing on-line Statistical Machine Learning (SML) techniques such as fast-incremental-model tree and sliding window random forest. The real time engine uses a proactive approach retraining the model periodically in order to overcome the concept drift issue. In the validation stage, interleaved test-train was introduced for online learning and the experiments showed an accuracy of 14% in terms of Normal Mean Absolute Error (NMAE) for video and audio and 28.5% for network read.

In [32] the concept of SLA conformance in terms of video frame and audio buffer rates was introduced as a classification problem focusing on improving the efficiency of SML techniques using distributed analytics based on local prediction models computed...
Related Work

on local stats and a global prediction model. The feature set was selected based on domain knowledge. Local models were built using classical base learners such as logistic regression, decision trees and random forest. The global model was develop using Winnow algorithm for model fusion. The model was evaluated based on classification and balanced accuracy, training time and network overhead, from which random forest trees presented the highest performance in terms of accuracy but a lower computational efficiency compared to the other SML methods. However, the distributed analytics approach showed better computational time efficiency and better scalability features when compared with a fully centralized approach, making it more appropriate for real time scenarios.

A study has also been conducted in order to generate initial prototypes for automated assurance systems using machine learning. A classification system for real-time prediction of SLA conformance or violation was developed based on the server cluster capacity region for a key-value store [3]. Additionally, a system prototype has been proposed to predict the resources (CPU, RAM) needed for a specific service in real time based on statistical machine learning [1].

The demonstrators accuracy showcases the feasibility of real-time prediction models under dynamic load patterns for real-time cloud capacity estimation based on service quality requirements and its load demand. However, additional work is still needed to generate more generic and adaptive models that can integrate multi-service and virtualized environments thus enabling end-to-end automated service assurance systems [31].

In [3], a real time SLA conformance prediction system was developed for a distributed key-value store (Voldemort) using the concepts of capacity and feasibility regions (load and resource spaces) in terms of read and write rates. The main objective was to predict SLA conformance or violation as a classification problem using an online setting.

In [33] a study has been conducted in order to generate an on-the-fly model for the system behavior using streaming device-level metrics as a way to predict client-side SLA violations. This work uses service-agnostic Online Machine Learning (OML), more specifically ascent logistic regression and Hoeffding Tree, and the results of testing the approach for a video-on-demand service under dynamic load patterns reflects a classification accuracy greater than 90% and less than 10% of false alarms, which confirms the appropriateness of online learning over offline mechanisms.

In [1], the research was more oriented towards resource prediction for cloud management following an anomaly detection approach. Based on the distributed video on demand service and considering different load patterns, resource usage prediction models were generated considering CPU and memory model based on service load (more specifically video frame and active TCP sessions).

In [34], a percentile-based performance analysis was developed for unreliable IaaS clouds. This approach considers SLA as performance metric based on the response time threshold having the conformance/violation rate as optimization constraint. This
problem is addressed as an optimal capacity decision problem for cost minimization under constraints of request rejection and SLA violation rates. The testbed is a real world scenario of an IaaS on top of OpenStack. Performance data was collected considering variable load load intensities, fault frequencies, multiplexing abilities and instantiation processing time (including faulty VM instantiation). The non-linear optimization problem was modeled using a simulated annealing method using the Metropolis algorithm. To evaluate the resource, comparisons between theoretical and empirical tail distributions were performed to evaluate the model correctness and accuracy.

In [35], an edge capacity planning solution was proposed for hierarchical edge cloud providing estimates of the minimum capacity needed in terms of the number of edge nodes under different loads for QoS compliance in terms of response delay, CPU/GPU demands, network bandwidth and latency. Furthermore, a smart combination of tasks was considered in order to increase resource utilization without incurring in QoS violations. The testbed includes edge computing considerations on hybrid clouds. The real time service used is an augmented virtual navigation and information system on docker containers. The case of study includes both simulated and real world experiments using Amazon EC2.

In [36], a regression-based resource estimation model was presented for VMs selection and provision with minimal configuration. The non-linear regression model generates provision estimations of key resources in order to meet workloads in different scales generated synthetically using Siege. The scenario consists of an experimental virtualized platform of a small cloud provider with different VM configurations running a web application. The testbed includes Virtual Resource Controller (VRC) (XenServer), a Provisioning Manager (PM) and a HTTP proxy and LB (Nginx). HTTP workload tests were used to estimate Virtual CPU (vCPU) demand, required RAM, server availability rate based on the percentage of replied and dropped requests. From these tests, an availability boundary was determined having the CPU as the main bottleneck factor. This method detected resource over-provisioning of 21% on maximum workload peak and 83% on average workload.

In [4], a specific case was studied using Hadoop MapReduce from a job-centric approach. The framework Bazaar was proposed as a way to predict how the performance of a MapReduce job scales in terms of multiple resources using a simple analytical model with job profiling. The users specify performance requirements for a given job in terms of completion time and/or desired cost and the most affordable resource combination for the cloud service provider is determined considering the given constraints. Based on the experimentation process, it is concluded that this framework makes fast predictions while keeping an average error lower than 12%. When outliers and failures appear, this framework finishes jobs earlier than expected by dedicating idle resources in order to fulfill user requirements despite imperfect predictions.

In [37] a solution was proposed in order to select the most suitable type of VM under dynamic loads when the system detects a need for it to scale horizontally considering
the performance variability among different VMs. This approach uses on-line machine learning in order to adapt the system in real-time under different workload patterns, application changes and middle-ware upgrades or reconfigurations. Results show that this approach is capable of automatically adapting the system to changes. Here, a Hierarchical Temporal Memory (HTM) is used in order to automatically detect changes in the application behavior. Several ANNs are trained in order to estimate the relationship between the number of served users and the resource usage considering CPU and RAM utilization. Both the learning rate and the momentum used for the ANN learning process are adjusted based on the anomaly score given by the HTM model so that the model is quickly adapted under different usage patterns. Based on that, the most suitable VM is selected by optimizing the ratio of minimal cost versus number of users. The approach was validated on the Amazon Elastic Compute Cloud (EC2) showing promising results.

In [38], a framework called Ernest has been implemented in order to predict performance in terms of running time for large scale analytics applications under different resource configurations so that the best configuration can be chosen in heterogeneous multi-tenant environments. The framework also chooses training points efficiently to provide accurate predictions with low overhead. This framework learns the performance behavior of the job execution based on small datasets and cluster configuration using simple models and is evaluated based on the performance on larger datasets and cluster setups on top of EC2. The evaluation shows prediction errors lower than 20% and training overheads lower than 5%. This study has shown that several advanced analytics workloads can be modeled using very simple feature sets related to their computation and communication patterns.

In [23], a novel SLO-based scheduler for Hadoop called Automated Resource Inference and Allocation (ARIA) was implemented. This scheduler is able to estimate the amount of resources to be allocated so that certain constraints i.e. job completion deadlines are met. This scheduler is built upon job profiles giving information about the performance of the job during all map, shuffle, sort and reduce stages. It also uses a MapReduce performance model based on a job, its profile, its input data and its defined SLO (soft deadline) to determine the amount of map and reduce slots needed to fulfill the job completion constraint using an Earliest Deadline First (EDF) policy. This scheduler was tested on a 66-node Hadoop cluster and the results show predicted completion times within 15% of the real times thus effectively meeting the job’s SLO until the demands exceed the available resources.

In [24], an Adaptive Scheduler (AS) for Hadoop was implemented capable of determining, based on a submitted job and its required SLA, if a job can be executed meeting the requirements considering the available resources. If it is capable, the job is scheduled, otherwise a new SLA is negotiated with the user. The SLA helps the AS to improve the job scheduling process. This study demonstrates the advantages of this approach over more classical scheduling approaches.
In [39], a framework was developed for the estimation and management of resources based on machine learning for video analytic applications on top of Hadoop given certain performance constraints in order to provide a QoS at a low cost. The job execution time is modeled and predicted considering parameters such as video file features, cluster resource consumption and Hadoop configuration values. The performance of several machine learning models were compared where the decision tree-based model for regression and the ensemble classifier outperforms other approaches.

In [40], a study was carried out in order to estimate resource utilization from data stream processing workloads in data centers ensuring guarantees on the QoS provided. A novel methodology has been used based on Mixture Density Networks (MDN), which combines ANN with mixture models in order to estimate the whole resource usage spectrum as PDF. The impact caused by the variability of the query behavior has been thus taken into account. These models were validated using linear road benchmark and the TCP-H, showing high accuracy even when there are changes in the distribution model of the processed queries based on several performance metrics such as MSE, continuous ranked probability score and negative log predictive density.

In [41], in order to contribute towards energy-efficient cloud systems, a load forecasting method was proposed using SML, more specifically Support Vector Regression (SVR) and a Kalman smoother, showing very good performance for resource estimation tasks based on its accuracy, stability and adaptability even under high variability when comparing with other approaches such as ANN and standard SVR. The predicted results are used for computing resource provisioning strategies, capable of saving till 48% of the resource consumption while ensuring SLAs.

In [5], a study was carried out proposing a model for workload forecasting used for optimal resource allocation. Results show how the model helps satisfying the QoS while keeping operational costs low.

Chapter 6 will discuss the connection of some of this prior art with the contribution carried out with the development of this project.
4 Methodology

The system prototype proposed in this project is shown in Fig. 4.1 whose components will be explained in the following sections.

![Figure 4.1: System Prototype.](image)

4.1 Testbed Set-up

4.1.1 Hardware Resource Identification

For the development of this project, access to 5 physical machines belonging to SICS North Data Center (DC) were provided by Ericsson. Each of this super-computers has the following specifications:

- Number of CPUs: 24
- Memory: 264 GB
- Network Interface: 10 Gbps
- Storage: 1 TB
- Operating System: CentOS

4.1.2 Cloud Service Decision

Table 4.1 shows the comparison among different services, taking into account:

- System components such as server, client, load generator, LB, etc.
- Available tools for sensing benchmarking and stressing the service.
• Possible model inputs/outputs in terms of features, KPIs and SLA.

After considering the different options, Hadoop MapReduce framework is selected as the PaaS to be used for the development of this project due to the its current relevance in the field of Big Data processing specially generated from IoT, for the previous efforts to develop estimation models of the platform performance and for all the tools that are available for benchmarking, stressing and monitoring.

4.1.3 Testbed Design and Deployment

System Architecture Design

Fig. 4.2 shows the system architecture design for Hadoop MapReduce service made up of a master node and 4 slave nodes, each of them comprising several micro-services.

Master Node

1. YARN resource manager on port 80880
2. HDFS name node on port 50070
3. Map-Reduce (MR) history server on port 8188

Slave Node

1. Node manager on port 8042
2. Data node on port 50075

Each micro-service has been deployed using a Docker container as a light weight virtualization option. Docker Swarm is used as a way to automatically manage the nodes in the cluster. The swarm’s manager is configured as the Hadoop master node and the swarm’s workers as the Hadoop slaves.

Hadoop MapReduce service version 2.7.4 is deployed based on the GitHub repository for the project Big Data Europe: Docker Apache HBase [42] using only the stack defined for Apache Hadoop on the distributed set-up. This repository uses images available from the Docker repository bde2020 for each micro-service. This repository belongs to the project Big Data Europe 2020 intended to provide Hadoop-as-a-Service.
Table 4.1: Service Comparison

<table>
<thead>
<tr>
<th>Service</th>
<th>Server-side Storage</th>
<th>Client-side Storage</th>
<th>Server-side Data Processing</th>
<th>Client-side Data Processing</th>
<th>Server-side Sensors</th>
<th>Client-side Sensors</th>
<th>Server-side Service KPI(s)</th>
<th>Client-side Service KPI(s)</th>
<th>Benchmarking / Stress Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Hadoop</td>
<td>HDFS</td>
<td>Hadoop</td>
<td>MapReduce</td>
<td>Hadoop</td>
<td>GW</td>
<td>YARN</td>
<td>Throughput</td>
<td>Throughput</td>
<td>CPU</td>
</tr>
<tr>
<td>MongoDB</td>
<td>-</td>
<td>Doc Store</td>
<td>Doc Client</td>
<td>Mongo DB</td>
<td>-</td>
<td>-</td>
<td>CPU</td>
<td>CPU</td>
<td>RAM</td>
</tr>
<tr>
<td>Video Streaming</td>
<td>Transcoding Instance</td>
<td>Gluster FS</td>
<td>VLC Client</td>
<td>HTTP LB</td>
<td>Streaming</td>
<td>HTTP LB</td>
<td>CPU</td>
<td>CPU</td>
<td>Memory</td>
</tr>
<tr>
<td>Voldemort</td>
<td>-</td>
<td>-</td>
<td>Voldemort Benchmark</td>
<td>Voldemort Benchmark</td>
<td>-</td>
<td>-</td>
<td>CPU</td>
<td>CPU</td>
<td>Memory</td>
</tr>
<tr>
<td>WebApp</td>
<td>Apache Web Server</td>
<td>-</td>
<td>HTTP Client</td>
<td>NGINX</td>
<td>-</td>
<td>-</td>
<td>CPU</td>
<td>CPU</td>
<td>Benchmark</td>
</tr>
</tbody>
</table>

© Uppsala University

Aleksandra Obeso Duque
Services are configured using *global mode*, so every time a new worker node joins the swarm, Hadoop slave’s micro-services are automatically deployed on that node.

**Hadoop Configuration**

Hadoop environment has been configured with the following main parameters:

- YARN scheduler: Capacity scheduler
- HDFS replication factor: 1
- HDFS block size: 128 MB
- Node manager resource CPU: 24 vCores
- Node manager resource memory: 264000 MB

### 4.2 Experimental Design

Three important points regarding resource utilization and service throughput\(^1\) are taken into account at the time of defining the experiments for data collection:

- The input data size
- The job complexity
- The cluster set-up (as the number of slave nodes)

For the purpose of delimiting this project’s scope, it will be assumed that single MapReduce jobs are running at the same time (as already mentioned, each job is made up of several map/reduce tasks running in a parallel and distributed way).

The input data size was chosen based on the block size configured in Hadoop so that full capacity per task is defined. In this case, the block size is 128MB and then based on that the set of input data vs. the number of map tasks generated by the framework are:

- 128 MB - 1 map task
- 256 MB - 2 map tasks
- 512 MB - 4 map tasks
- 1 GB - 8 map tasks
- 2 GB - 16 map tasks
- 4 GB - 32 map tasks
- 8 GB - 64 map tasks

\(^1\)Throughput as the amount of data that is processed per unit time measured in MB/s
• 16 GB - 128 map tasks
• 32 GB - 256 map tasks

On the other hand, for the project’s delimitation, the number of reduce tasks is kept constant. Also, the data-locality was kept constant so that the data is stored in the same machine along all experiments. This way effects of having data replicated in different nodes is not considered and should be further studied.

Based on the Hadoop MapReduce pipeline, the following two benchmarking tools are considered during the experiments. This tools are available in the Hadoop MapReduce examples JAR file.

• Wordcount (WC): A MapReduce program that counts the words in the given input files.
• Terasort (TS): A MapReduce program that sorts the data written by the random writer TeraGen.

Fig. 4.3 showcases how WC and TS put stress on different stages of the Hadoop MapReduce framework. These examples belong to the case when having an input data size of 4 GB and 1 Master - 3 Slave (1M3S) as the cluster set-up; the job is split into 32 map tasks. As can be deduced from the two profiles, WC (left) puts a higher stress on the map stage whereas TS (right) stresses more the shuffle-sort stage. WC is a very well-known tool to test the computational power, whereas TS is good to evaluate the system’s behavior in terms of the networking capabilities.

The data used as input for each benchmarking tool are:

• For WC, a text file containing the e-book *Don Quijote*, by Miguel de Cervantes Saavedra, is used as the base file with a size of 2 MB from which larger input files were generated by data replication.
• For TS, the tool TeraGen is used to generate random input data.
The different cluster set-ups are:

- 1 Master - 1 Slave (1M1S)
- 1 Master - 2 Slave (1M2S)
- 1 Master - 3 Slave (1M3S)
- 1 Master - 4 Slave (1M4S)

### 4.3 Data Extraction Pipeline

#### 4.3.1 Data Collection and Parsing

Fig. 4.4 shows the data extraction pipeline comprising collection, parsing and identification of features at the infrastructure level and the definition of targets at the platform level.

![Data Extraction Pipeline](image)

**Figure 4.4: Data Extraction Pipeline.**

**Infrastructure-level Utilization Statistics**

SAR is used in order to monitor each node’s usage statistics of the infrastructure-level resources. It is configured so that data is collected every second. The Network Time Protocol (NTP) is used in order to synchronize the clock of all the system’s nodes.

The statistics collected with this tool are:

- CPU stats
Methodology

- Memory utilization stats
- Network stats
- Disk I/O and transfer rate stats

SADF tool [43] allows to parse the data collected by SAR into different formats. In this project, it is used to parse the SAR stats into JSON format.

Platform-level KPIs

The JHist event handler [44] is a service available in the framework itself and may be used in order to profile the MapReduce service performance. These job history events are stored in the HDFS. It is a task-based tool in charge of collecting important events and for each of them a timestamp is added. These files are stored using an Avro Json format [45] and include the following information:

- Type of event
- Number of map and reduce tasks
- Execution times (submit, launch and finish times)
- Read and written bytes (HDFS and local file system)
- Number of read and write operations (HDFS and local file system)
- Input and output number of records
- Job status

Hadoop Rumen - Trace Builder [46] is a tool that allows to parse JHist logs. This tool also uses Cumulative Distribution Functions (CDF) in order to extrapolate incomplete information. This tool is used then to convert the JHist logs into JSON format.

4.3.2 Model’s I/O Definition

Feature Identification

After extracting around 370 different infrastructure-level statistics provided by SAR, approximately 53 different metrics per node are selected from which, the ones belonging to the slave nodes were aggregated using an average. The aggregation is required so that a general model can be inferred and used no matter the number of slave nodes in the cluster. The set resulting from concatenating the master node’s features and the aggregated slave nodes features represent the full set of features.
KPIs Definition

The target metrics are defined based on the Hadoop MapReduce performance metrics collected by JHist. As explained before, these performance metrics are collected on a task-based granularity and based on the events triggered by the framework’s execution. These metrics are matched time-wise and aggregated from all the tasks running in a parallel and distributed way. From them, an estimation of the service throughput or Carried Load (CL) is derived representing the amount of information that can be processed in the cluster per unit time. So, the platform-level KPIs are defined as:

- CL Map: Map stage throughput measured in \( MB/s \).
- CL Reduce: Reduce stage throughput measured in \( MB/s \) (includes the shuffle and sort stages since JHist does not provide finer metrics regarding the amount of data processed exclusively by the reduce stage, at least when specifying only 1 reduce task).

SLA Definition

The SLA is defined in terms of the MapReduce throughputs so that it can be used as an indicator of when the number of nodes shall be increased if the violation flag is raised and hold for a period of time.

In order to develop a classification model for the automatic detection of SLA violation, raw SLA thresholds were chosen for each KPI and benchmark. Each KPI has a different threshold since the CL Reduce indicator includes the shuffle and sort stages which have a different behavior due to the intercommunication of the data. Furthermore, for this project delimitation, the number of reduce tasks is kept constant along the experiments, which is not the case for the map stage.

The binary classification is then defined as:

Table 4.2: SLA definition.

<table>
<thead>
<tr>
<th>SLA Map</th>
<th>SLA Red</th>
<th>Explanation</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Fulfillment</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>SLA Red Violation</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>SLA Map Violation</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>SLA Map and Reduce Violation</td>
<td>1</td>
</tr>
</tbody>
</table>

2Note that the job execution time is not considered as a KPI in this project since the idea is to infer models that can enable proactive decision making by using online approaches and since MapReduce is mainly used to process large amounts of data, this processing takes considerably long time. The option of using small input data would result in shorter execution times but due to the fact that the number of map tasks spawned by the framework depends on the relation input data size vs. HDFS block-size, this approach would require to trick this parameter for each experiment run and the service re-deployment, which would be more time-consuming.

3A more rigorous study shall be performed to better tune the thresholds and also for the opposite scenario of when the resources should be reduced.

4It would be needed to carried-out a more rigorous study of the service’s KPI density function to come up with more meaningful thresholds.
A different threshold for each experiment case was define as shown in Table 4.3.

Table 4.3: SLA example thresholds.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Cluster Setup</th>
<th>Threshold Map [MB/s]</th>
<th>Threshold Red [MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC</td>
<td>1M1S</td>
<td>50</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>75</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>125</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>160</td>
<td>0.22</td>
</tr>
<tr>
<td></td>
<td>1M1S</td>
<td>100</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>125</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>150</td>
<td>5.00</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>170</td>
<td>5.00</td>
</tr>
</tbody>
</table>

4.4 Tools for Data Analytics

For both pre-processing the data, training and validation of the ML-based models, algorithms are developed using Python and more specifically using the libraries Scikit Learn for ML and Pandas for data analytics based on dataframes. The following sections detail the process carried out for both pre-processing and generating the regression and classification models.

4.5 Generation of I/O Traces

Fig. 4.5 shows the trace generation pipeline including both the SAR and the JHist interpreters, more detailed in the next subsections.

Figure 4.5: Trace Generation Pipeline.

4.5.1 SAR Interpreter

For each experiment the following steps were considered to generate SAR traces:
Methodology

- Extract CPU, I/O, Memory and Network usage statistics
- Interpolate missing samples in each metric
- Match with CL traces and remove SAR samples out of experiments
- Aggregate slave metrics using average

4.5.2 JHist Interpreter

The following steps were considered in order to generate the CL traces for MapReduce only considering successful job profiles collected by JHist:

- Filter out failed jobs
- Extract event-based metrics
- Estimate average CL per task
- Generate KPI traces aggregating the average CL of all tasks running in parallel at each time step (1s)

4.6 Pre-processing

Fig. 4.6 shows the different pre-processing techniques applied to the collected traces generating different datasets, the natural dataset, the over-sampled dataset, the training/test datasets and the reduced dataset. The different stages are explained in more detail in the following subsections.

4.6.1 Traces Concatenation

After generating traces for each experiment, they are concatenated in time to produce longer traces for each feature and each target correlated in time.
4.6.2 Feature Selection

The feature selection is performed combining some prior-knowledge in order to keep only numeric SAR data and also applying variance threshold on the resulting average features from concatenating all experiments so that features with very low variance, including constant features, are excluded from the analysis. In this way, the feature set was reduced to a total of 73 concatenated features, from which 38 are from the master node and 35 aggregated features from the slave nodes. The resulting dataset from these traces is called natural dataset.

4.6.3 Over-sampling

Each experiment scenario has produced traces with different time duration thus resulting in a different number of samples per experiment. Also, setting SLA thresholds has produced different number of samples for violation and fulfillment. To avoid any bias caused by class-imbalance that may prevent to get a good general model accurate enough for all the cases, random over-sampling with replacement has been applied in such a way that the minority classes can have the same number of samples than the majority class.

<table>
<thead>
<tr>
<th>No. Samples</th>
<th>1M1S</th>
<th>1M2S</th>
<th>1M3S</th>
<th>1M4S</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC</td>
<td>2075</td>
<td>1666</td>
<td>1242</td>
<td>1243</td>
</tr>
<tr>
<td>TS</td>
<td>1179</td>
<td>4916</td>
<td>4614</td>
<td>4578</td>
</tr>
</tbody>
</table>

Table 4.4: Samples Collected per Experiment: Imbalance in Regression.

<table>
<thead>
<tr>
<th>Class</th>
<th>No. Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLA Violation</td>
<td>5757</td>
</tr>
<tr>
<td>SLA Fulfillment</td>
<td>15756</td>
</tr>
</tbody>
</table>

Table 4.5: Samples per SLA class: Imbalance in Classification.

This methodology is applied for both regression and classification problems. In the case of regression the classes make direct reference to each experiment scenario (see Table 4.4). On the other hand, for the case of classification, classes make reference to the binary classes defined in Section 4.3.2 (see Table 4.5). The resulting dataset after applying over-sampling is called over-sampled dataset.

4.6.4 Data Split

For both the regression and classification targets, the natural dataset is split into 70% for train dataset and 30% for test dataset using stratification following the same class definition mentioned in the previous subsection for regression and classification. Stratification allows to avoid class imbalance in the resulting datasets.
4.6.5 Feature Scaling

Minmax scaling for each feature is applied in order to keep the same relevance among all features.

4.6.6 Dimension Reduction

In order to avoid the curse of dimensionality on this problem, PCA is applied to the 73 features in order to keep 99% of the variance having then 23 features. The resulting dataset is called reduced dataset.

4.7 Batch Learning

Models generated using batch learning are developed and compared for each of the following data sets, applying both regression and classification approaches shown in Fig. 4.7.

- Natural dataset
- Over-sampled dataset
- Reduced dataset
- Over-sampled and reduced dataset

![Figure 4.7: Machine Learning Approaches for Regression and Classification.](image)

The training process is carried out on testbed nodes.

4.7.1 Regression

As mentioned before, the regression problem is defined in the way that platform-level KPIs $\hat{Y}_{\text{CarriedLoadSpace}}(\text{CLS})$ (also known as CL space) can be deduced from infrastructure-level utilization statistics $\hat{X}_{RS}$ (also known as resource space). As specified next, different regression models are trained and tested in order to compare their performance to deduce the best approach for this particular problem. For validation, metrics such as $R^2$,
%NMAE and training time are used to compare the models performance based on the test dataset.

**MLP Regressor**

In order to tune hyper-parameters, the following cases are considered at the time of training the Multi-layer Perceptron (MLP)-based regression model:

- Solvers: [SGD, Adam]
- Configuration of hidden layers with respect to the number of layers and number of nodes in each layer: (20), (20, 15), (20, 15, 10) (20, 15, 10, 5)
- Early stop: [True, False]
- Activation function: [logistic, tanh]
- Learning rates: [Constant, adaptive]
- Number of models (as the number of models generated after training with different parameter’s initialization): 5

**Lasso Regressor**

To tune hyper-parameters, the following cases are considered at the time of training the lasso-based regression model:

- Alphas: [10, 5, 2, 1, 0.8, 0.5, 0.1, 0.001, 0.0005, 0.0001]
- Selection: [random, cyclic]
- Number of models: 5

**RF Regressor**

For hyper-parameter tuning, different configurations are considered when training the Random Forest (RF) regression model:

- Number of estimators: [1, 10, 100, 250, 500]
- Number of models: 5

### 4.7.2 Classification

The classification problem is defined in the way that SLA violation/fulfillment $\hat{Y}_{\text{SLA}}$ (based on the CL space) can be detected from infrastructure-level utilization statistics $\hat{X}_{\text{RS}}$ (also known as resource space). Three different classification models are trained and tested in order to compare their performance to deduce the best approach for this particular problem. For validation, metrics such as accuracy, confusion matrix and training time are used to compared the models performance based on the test dataset.
**MLP Classifier**

In order to tune hyper-parameters, the following cases are considered at the time of training the MLP-based regression model:

- Solvers: [Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (LBFGS), SGD, Adam]
- Configuration of hidden layers with respect to the number of layers and number of nodes in each layer: (20), (20, 15), (20, 15, 10) (20, 15, 10, 5)
- Early stop: [True, False]
- Activation function: [logistic, tanh, identity, relu]
- Learning rates: [Constant, adaptive, inverse scaling]
- Number of models: 5

**Ridge CV Classifier**

To tune hyper-parameters, the following cases are considered at the time of training the lasso-based regression model:

- Alphas (regularization strength): [10, 5, 2, 1, 0.8, 0.5, 0.1, 0.001, 0.0005, 0.0001]
- Number of models: 5

**RF Classifier**

For hyper-parameter tuning, different configurations are considered when training the RF regression model:

- Number of estimators: [1, 10, 100, 250, 500]
- Split criteria: [gini coefficient, entropy]
- Number of models: 5
5 Results

5.1 System Resource Utilization

This section is intended to compare in a more detailed way how the two kinds of benchmarking tools, Wordcount (WC) and Terasort (TS), stress the resources at the infrastructure level. The following figures showcase the Resource Utilization (RU) at each node for a cluster set-up of 1M3S\(^1\) and an input data size of 32 GB (256 map tasks) in terms of computation 5.1, memory 5.2, network 5.3 and disk I/O 5.4 collected from the SAR stats.

![Figure 5.1: % CPU Utilization for 1M3S and 32 GB.](image)

![Figure 5.2: % Memory Utilization for 1M3S and 32 GB.](image)

\(^{1}\): Master, 2: Slave-1, 3: Slave-2, 4: Slave-3
Results

Figure 5.3: % Network Utilization for 1M3S and 32 GB.

(a) WC  (b) TS.

Figure 5.4: Disk I/O Utilization (TPS) for 1M3S and 32 GB.

(a) WC  (b) TS.

These results match the expected behavior. The WC benchmark stresses the CPU capabilities (Fig. 5.1) whereas the TS benchmark stresses more the networking capabilities (Fig. 5.3) and at the same time stresses the disk I/O in terms of Transactions per Second (TPS) enforcing the bottleneck during the shuffling stage.

5.2 System Performance

This section is intended to show the scalability of the distributed testbed considering the different experimental scenarios. The following figures show how the distributed system performs when changing the input data size, the complexity of the MapReduce job with the two benchmarks and the cluster set-up.
Figure 5.5: Input Data Size vs. Elapsed Time for different Cluster Set-ups.

Figure 5.6: Input Data Size vs. Max. Map Throughput for different Cluster Set-ups.

Figure 5.7: Shuffle Data Size vs. Max. Shuffle-Sort-Reduce Throughput for different Cluster Set-ups.

From the graphs above, the scalability of the system can be evaluated, specially for the WC benchmark. For example, with 2 slave nodes, the map stage can handle around $24 \times 2 = 48$ tasks in parallel (6 GB for a block size of 128 MB). The scalability starts
Results

degraded after this threshold, as seen in Fig. 5.6. From Fig. 5.7 the throughput of the shuffle, sort and reduce has been plotted showing a very low performance in terms of scalability due to the fact that the number of reducers was kept unitary along all experiments, but specially since the shuffle stage represents a bottleneck in the framework.

5.3 Resource Utilization and Carried Load Traces

Fig. 5.8 shows both the resource utilization and carried load traces collected while running WC benchmark for each cluster set-up and varying the input data size. Fig. 5.9 shows the same traces for TS benchmark.

5.4 Resource Utilization and Carried Load Spaces

Fig. 5.10 shows both the resource utilization and carried load spaces for the collected data from running WC benchmark for each cluster set-up and varying the input data size. Fig. 5.11 shows the same spaces for TS benchmark.
Figure 5.8: RU and CL Traces when varying Input Data Size: WC.
Figure 5.9: RU and CL Traces when varying Input Data Size: TS
Figure 5.10: RU and CL Spaces: WC.
Figure 5.11: RU and CL Spaces: TS.
5.5 KPI Data Distribution

The following figures show the PDF of the KPIs for all experimental scenarios, more specifically Fig. 5.12 shows the different PDFs for WC benchmark, whereas Fig. 5.13 the different PDFs for TS benchmark.

Figure 5.12: PDF generated using KDE: WC.
Figure 5.13: PDF generated using KDE: TS.
5.6 Batch Learning

5.6.1 Regression

MLP Regressor

From all the different MLP regressors that were trained to tune the hyper-parameters, the model that performs the best was trained with the following parameters and its performance on different experiment scenarios is shown in Table 5.1.

- Training dataset: Natural
- Solver: LBFGS
- Hidden layer configuration: (20, 15)$^2$
- Early stop: True
- Activation function: Logistic
- Learning rate: Adaptive

Table 5.1: Performance: MLP Regressor.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Cluster Set-up</th>
<th>CL Map</th>
<th>CL Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark</td>
<td>Mean$^3$</td>
<td>$R^2$</td>
<td>%NMAE</td>
</tr>
<tr>
<td>WC</td>
<td>1M1S</td>
<td>33.841</td>
<td>0.912</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>43.973</td>
<td>0.980</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>52.972</td>
<td>0.992</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>56.778</td>
<td>0.991</td>
</tr>
<tr>
<td>TS</td>
<td>1M1S</td>
<td>11.469</td>
<td>0.978</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>17.565</td>
<td>0.994</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>15.128</td>
<td>0.996</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>15.0603</td>
<td>0.994</td>
</tr>
<tr>
<td>Overall</td>
<td>24.102</td>
<td>0.993</td>
<td>6.432</td>
</tr>
</tbody>
</table>

Fig. 5.14 shows how this MLP regressor performs when predicting the CL traces vs. the expected ones for WC benchmark whereas Fig. 5.15 shows the traces for TS benchmark.

$^2$Number of neurons in each hidden layer
Figure 5.14: Expected vs. Predicted CL using MLP Regression: WC.
Figure 5.15: Expected vs. Predicted CL using MLP Regression: TS.
Lasso Regressor

From all the different Lasso regressors that were trained to tune the hyper-parameters, the model that performs the best was trained with the following parameters and its performance on different experiment scenarios is resumed in Table 5.2.

- Training dataset: Natural
- Alphas: 0.0001
- Selection: Cyclic

Table 5.2: Performance: Lasso Regressor.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Cluster Set-up</th>
<th>CL Map Mean</th>
<th>CL Map $R^2$</th>
<th>CL Map %NMAE</th>
<th>CL Red Mean</th>
<th>CL Red $R^2$</th>
<th>CL Red %NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC</td>
<td>1M1S</td>
<td>33.841</td>
<td>-1.557</td>
<td>55.352</td>
<td>0.040</td>
<td>-159.672</td>
<td>1817.660</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>43.973</td>
<td>0.435</td>
<td>33.361</td>
<td>0.058</td>
<td>-95.629</td>
<td>923.437</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>52.972</td>
<td>0.776</td>
<td>29.163</td>
<td>0.073</td>
<td>-75.613</td>
<td>972.014</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>56.778</td>
<td>0.839</td>
<td>27.989</td>
<td>0.0799</td>
<td>-277.689</td>
<td>1917.750</td>
</tr>
<tr>
<td>TS</td>
<td>1M1S</td>
<td>11.469</td>
<td>0.762</td>
<td>68.983</td>
<td>4.089</td>
<td>-0.084</td>
<td>35.136</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>17.565</td>
<td>0.881</td>
<td>46.7077</td>
<td>3.944</td>
<td>0.372</td>
<td>16.906</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>15.128</td>
<td>0.861</td>
<td>59.080</td>
<td>4.231</td>
<td>0.087</td>
<td>17.658</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>15.060</td>
<td>0.789</td>
<td>72.039</td>
<td>4.275</td>
<td>0.274</td>
<td>15.069</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>24.102</td>
<td>0.817</td>
<td>46.828</td>
<td>2.960</td>
<td>0.726</td>
<td>26.012</td>
</tr>
</tbody>
</table>

Fig. 5.16 shows how this Lasso regressor performs when predicting the CL traces vs. the expected ones for WC benchmark. Fig. 5.17 shows the traces comparison for TS benchmark.
Figure 5.16: Expected vs. Predicted CL using Lasso Regression: WC.
Figure 5.17: Expected vs. Predicted CL using Lasso Regression: TS.

**RF Regressor**

From all the different RF regressors that were trained to tune the hyper-parameters, the model that performs the best was trained with the following parameters and its performance on different experiment scenarios is resumed in Table 5.3.

- Training dataset: Over-sampled
- Number of estimators: 100
Table 5.3: Performance: RF Regressor.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Cluster Set-up</th>
<th>CL Map Mean</th>
<th>CL Map R²</th>
<th>CL Map %NMAE</th>
<th>CL Red Mean</th>
<th>CL Red R²</th>
<th>CL Red %NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>WC</td>
<td>1M1S</td>
<td>33.300</td>
<td>0.997</td>
<td>0.589</td>
<td>0.047</td>
<td>0.950</td>
<td>8.337</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>41.941</td>
<td>0.998</td>
<td>0.818</td>
<td>0.056</td>
<td>0.927</td>
<td>10.314</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>55.398</td>
<td>1.000</td>
<td>0.453</td>
<td>0.072</td>
<td>0.963</td>
<td>7.767</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>55.243</td>
<td>1.000</td>
<td>0.450</td>
<td>0.073</td>
<td>0.965</td>
<td>7.834</td>
</tr>
<tr>
<td>TS</td>
<td>1M1S</td>
<td>13.868</td>
<td>0.999</td>
<td>1.089</td>
<td>4.054</td>
<td>0.997</td>
<td>0.566</td>
</tr>
<tr>
<td></td>
<td>1M2S</td>
<td>16.936</td>
<td>0.995</td>
<td>4.295</td>
<td>4.052</td>
<td>0.929</td>
<td>2.086</td>
</tr>
<tr>
<td></td>
<td>1M3S</td>
<td>16.473</td>
<td>0.996</td>
<td>4.270</td>
<td>4.244</td>
<td>0.905</td>
<td>1.833</td>
</tr>
<tr>
<td></td>
<td>1M4S</td>
<td>14.691</td>
<td>0.997</td>
<td>3.819</td>
<td>4.255</td>
<td>0.908</td>
<td>1.884</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>30.979</td>
<td>0.998</td>
<td>1.284</td>
<td>2.107</td>
<td>0.990</td>
<td>1.700</td>
</tr>
</tbody>
</table>

Fig. 5.18 shows how this RF regressor performs when predicting the CL traces vs. the expected ones for WC benchmark. Fig. 5.19 shows the traces comparison for TS benchmark.
Figure 5.18: Expected vs. Predicted CL using RF Regression: WC.
Figure 5.19: Expected vs. Predicted CL using RF Regression: TS.

5.6.2 Classification

MLP Classifier

From all the different MLP classifiers that were trained to tune the hyper-parameters, the model that performs the best was trained with the following parameters and its performance on different experiment scenarios is shown in Table 5.4. Metrics are detailed for each of the two classes predefined.
• Training dataset: Over-sampled and reduced

• Solver: LBFGS

• Hidden layer configuration: (20, 15, 10)

• Early stop: False

• Activation function: Tanh

• Learning rate: Adaptive

Table 5.4: Performance: MLP Classifier (Note that there are two values per metric, one per class).

<table>
<thead>
<tr>
<th>Accuracy Score</th>
<th>Train Time [s]</th>
<th>Confusion Matrix</th>
<th>Precision</th>
<th>Recall</th>
<th>F-β Score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.994</td>
<td>25.634</td>
<td>4674 53 8 4719</td>
<td>0.999</td>
<td>0.989</td>
<td>0.994</td>
<td>4727</td>
</tr>
</tbody>
</table>

Fig. 5.20 shows how this MLP classifier performs when predicting the SLA for all experimental scenarios (all input data sizes, all benchmarks and all cluster set-ups) with respect to the expected predictions on the test dataset.

Figure 5.20: Expected vs. Predicted Classification using MLP: c. Note that the red points denote SLA Violation whereas green denote SLA compliance

Ridge CV Classifier

From all the different Ridge Cross-Validation (CV) classifiers that were trained to tune the hyper-parameters, the model that performs the best was trained with the following parameters and its performance on different experiment scenarios is resumed in Table 5.5. Metrics are detailed for each of the two classes predefined.

• Alpha: 0.0001
Table 5.5: Performance: Ridge CV Classifier.

<table>
<thead>
<tr>
<th>Accuracy Score</th>
<th>Train Time [s]</th>
<th>Confusion Matrix</th>
<th>Precision</th>
<th>Recall</th>
<th>F-β Score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.835</td>
<td>0.245</td>
<td>3667 1060 502 4225</td>
<td>0.880</td>
<td>0.799</td>
<td>0.824</td>
<td>4727</td>
</tr>
</tbody>
</table>

Fig. 5.21 shows how this Ridge CV classifier performs when predicting the SLA for all experimental scenarios (all input data sizes, all benchmarks and all cluster set-ups) with respect to the expected predictions on the test dataset.

![Expected SLA: Test dataset](image1)

![Predicted SLA: Test dataset](image2)

Figure 5.21: Expected vs. Predicted Classification using Ridge CV: SLA.

RF Classifier

From all the different RF classifiers that were trained to tune the hyper-parameters, the model that performs the best was trained with the following parameters and its performance on different experiment scenarios is resumed in Table 5.6. Metrics are detailed for each of the two classes predefined.

- Training dataset: Over-sampled
- Number of estimators: 500
- Split criteria: Entropy

Table 5.6: Performance: RF Classifier.

<table>
<thead>
<tr>
<th>Accuracy Score</th>
<th>Train Time [s]</th>
<th>Confusion Matrix</th>
<th>Precision</th>
<th>Recall</th>
<th>F-β Score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.999</td>
<td>34.484</td>
<td>4718 9 3 4724</td>
<td>0.999</td>
<td>0.998</td>
<td>0.999</td>
<td>4727</td>
</tr>
</tbody>
</table>

Fig. 5.22 shows how this RF classifier performs when predicting the SLA for all experimental scenarios (all input data sizes, all benchmarks and all cluster set-ups) with respect to the expected predictions on the test dataset.

© Uppsala University

Aleksandra Obeso Duque
Figure 5.22: Expected vs. Predicted Classification using RF: SLA.
6 Discussion & Analysis

6.1 Model Comparison

6.1.1 Regression

Table 6.1 shows the performance comparison among the different models generated by using MLP, Lasso and RF regression. As it can be seen, the model that performs the best against the test data set is the RF-based with a %NMAE of 1.284% for CL Map and 1.700% for CL Reduce. Although a comparison among the different RF regressors using the different options of input datasets is not shown in this report, it can be said that the one that outperforms is the model trained with the full set of features (73) and using the over-sampling strategy so that it can perform well for all experimental scenarios. Although, in terms of training time, the RF regressor is the fastest to converge during the training process. On the other hand, the MLP regressor does not perform as well as the RF regressor but significantly better than the Lasso regressor. The MLP model is not able to predict CL Reduce as shown in Table 5.1. For all the cases the KPI that is hardest to predict is the CL Reduce, probably due to the trace’s mean value that is very low.

Table 6.1: Regressors comparison.

<table>
<thead>
<tr>
<th>Model</th>
<th>Over-Sampling</th>
<th>PCA</th>
<th>Train $R^2$</th>
<th>Train Time [s]</th>
<th>$R^2$</th>
<th>%NMAE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CL Map</td>
<td>CL Red</td>
</tr>
<tr>
<td>MLP</td>
<td>N</td>
<td>N</td>
<td>0.996</td>
<td>213.647</td>
<td>0.993</td>
<td>0.887</td>
</tr>
<tr>
<td>Lasso</td>
<td>N</td>
<td>N</td>
<td>0.820</td>
<td>115.449</td>
<td>0.817</td>
<td>0.726</td>
</tr>
<tr>
<td>RF</td>
<td>Y</td>
<td>N</td>
<td>1.000</td>
<td>6.308</td>
<td>0.998</td>
<td>0.990</td>
</tr>
</tbody>
</table>

6.1.2 Classification

Table 6.2 also shows the performance comparison among the different classifiers, detailing the different metrics per each of the two classes predefined. The one that performs the best is the RF-based classifier with an accuracy score of 99.9%, followed by the MLP classifier and then the Ridge CV classifier that is the one that performs the worst. However, in terms of training time, the RF-based classifier shows a longer training time before reaching convergence. In terms of the input dataset, keeping the full feature set allows in all cases to have a higher accuracy and applying over-sampling to cope with class-imbalance also helps to improve the classifier’s performance.
Table 6.2: Classifiers comparison.

<table>
<thead>
<tr>
<th>Model</th>
<th>Over-Sampling</th>
<th>PCA</th>
<th>Accuracy Score</th>
<th>Train Time [s]</th>
<th>Confusion Matrix</th>
<th>Precision</th>
<th>Recall</th>
<th>F-$\beta$ Score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>Y</td>
<td>N</td>
<td>0.994</td>
<td>25.634</td>
<td>4674 53</td>
<td>0.999</td>
<td>0.988</td>
<td>0.994</td>
<td>4727</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8 4719</td>
<td>0.989</td>
<td>0.998</td>
<td>0.994</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3667 1060</td>
<td>0.880</td>
<td>0.776</td>
<td>0.824</td>
<td>4727</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>502 4225</td>
<td>0.799</td>
<td>0.894</td>
<td>0.844</td>
<td></td>
</tr>
<tr>
<td>Ridge CV</td>
<td>Y</td>
<td>N</td>
<td>0.835</td>
<td>0.245</td>
<td>4718 9</td>
<td>0.999</td>
<td>0.998</td>
<td>0.999</td>
<td>4727</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 4724</td>
<td>0.998</td>
<td>0.999</td>
<td>0.999</td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>Y</td>
<td>N</td>
<td>0.999</td>
<td>34.484</td>
<td>4727</td>
<td>0.999</td>
<td>0.999</td>
<td>0.999</td>
<td></td>
</tr>
</tbody>
</table>

6.2 Challenges

One of the main goals of this project is to develop a methodology that can at least enable the use of online ML techniques in the future, to produce early predictions of SLA violation that can then trigger cloud service scaling processes on-time. In this sense, some of the aspects that were identified as challenging during the development of this project are, for example, the lack of performance sensors with a continuous output at the platform level. This is due to the fact that JHist, is an event-based tool that does not really produce continues performance metrics. As explained in Ch. 4, analysis of the event-based metrics was needed to produce an average estimation of the KPIs that could be correlated in time with the statistics collected by SAR. On the other hand, batch performance metrics such as the total execution time that are more understandable for the end-user at the application level, would make difficult to produce online predictions in a big data processing framework such as Hadoop, where usually jobs take a long time to be executed.

On the other hand, the need of generating prediction models based on ML posed some limitations in the definition of the feature set. More specifically, the need of defining a fix number of features despite the fact that in cloud environments the number of nodes changes in time as a way to cope with a dynamic load. For that, different aggregation strategies can be used; here a basic average was considered to tackle this issue and it showed to be enough for both regression and classification problems as shown in Figs. 5.18, 5.19 and 5.22. However, the use of other metrics such as percentile can be studied to take into account the effect of outliers.

Predicting MapReduce performance based on historical data of a particular job does not always hold due to the fact that the input data set can change in terms of size and content. Furthermore, the complexity of the MapReduce job can change drastically based on the interface implementation specified by the developer. If on top of that a dynamic distributed system such as a cloud environment that can change the amount of resources in time is added, the problem becomes even more complex.

Hadoop MapReduce itself is a framework with a large number of parameters that are specified before deployment and that can really impact the performance of the framework. For simplicity, many of these parameters were just kept constant. For
example, data locality would really affect the MapReduce workflow performance but it was left aside due to time constraints. On the other hand, due to the fact that light containers are used as virtualization, the cost of spawning new nodes has been neglected.

The main focus of this project was to evaluate the framework’s performance in terms of the input data size, different cluster set-ups and benchmark tools as a way to test the system behavior in terms of computational power and networking power. The computational power could be broadly tested changing the input data size which determines the number of map tasks to be spawn parallely. However, the networking capability of the cluster could not be broadly tested due to having data links with transmission rates of up to 10 Gbps and also due to the fact that the number of reducer tasks was kept constant thus producing a Disk I/O bottleneck as shown in Fig. 5.4.

Also for the definition of KPIs, it was complex to determine the performance metrics based on the metrics available in JHist. Since this library only gives information of the data size for the whole pipeline shuffle-sort-reduce, it was difficult to give finer metrics that allow a better analysis of the networking behavior of the Hadoop MapReduce framework.

The large number of features collected by SAR can introduce problems such as the curse of dimensionality discussed in Ch. 2. Also it was needed to take into account the class-imbalance for both regression and classification problems. In the regression problem, a bunch of experimental scenarios were defined and the data collected from each scenario had different time durations thus producing different number of samples that could cause bias in the inferred models towards the most representative scenario. Since the main goal of using batch learning was to generate a model good enough for all scenarios then over-sampling techniques were applied thus resulting in a improvement on the overall performance for the RF regressor model. Same case for the classification problem but in terms of the number of samples of each of the binary classes defined for SLA violation or fulfillment.

On the other hand, the definition of platform level SLA on a cloud environment was challenging in the sense that a rigorous analysis regarding the PDF is needed covering the different experimental scenarios based on the defined use case. The thresholds defined in Table 4.3 were generated considering Figs. 5.12 and 5.13. First, for each benchmark, a different SLA threshold was defined. For CL Map the threshold increases as it is expected the cluster to be able to handle a higher load. However, CL Reduce was kept constant since the number of reducers was kept constant along the experiments and also it showed very poor scalability when adding more nodes.

6.3 Connection with Previous Work

In order to connect the results obtained by the development of this project, the main differentiator with respect to previous projects developed at Ericsson and presented in Ch. 3, was the possibility to migrate to a more cloud-based environment where
considerations such as a variable number of nodes is taken into account during the data collection and the generation of ML-based models. In this sense, using a distributed system with light weight virtualization that could ease the resource scaling process was integrated. Also the use of a more cloud-related service such as Hadoop MapReduce for handling big data enabled the possibility to stress the available resources in the cluster. On the other hand, the evaluation of the results showed that it is possible to apply ML in such environment achieving very high accuracy.

Many of the works presented in the state of the art, evaluate the prediction of the SLA before the job is scheduled. However, in highly dynamic systems such as cloud-based, it would be very interesting to be able to monitor the service performance in real-time when it is being executed thus allowing to prevent any performance degradation experienced by the user.

The use case defined for this project had as a main goal to develop a methodology to enable the intelligent management of resources in a cloud environment by using prediction models as a way to prevent SLA violations. This early detection could enable the intelligent decision making for adapting the amount of available resources as the load changes. As a first step, having a self-aware cloud system capable of having knowledge of its performance at each node and able to map the behavior among the different service levels (IaaS, PaaS and SaaS), would allow the development of self-regulated cloud systems.

When comparing this first approach presented in this project against the possibility of having a connection from the infrastructure to the platform level via an API for example, this proposal would not represent a big advantage. However, the methodology developed here would enable as further work the development of more advanced ML mechanisms such as Long-Short Term Memory (LSTM) to provide even more opportune decision making. This solution, is also more appropriate when handling data stream processing such as the ones handled by Apache Spark when you do not know ahead how much data is going to be processed, which is the case for many real life scenarios where providing high performance is very critical.

On the other hand, this methodology can be paired with a model in charge of mapping from the platform level to the software level, in the case of monitoring the performance of the cloud system with respect to the end-user.
7 Conclusion

In this project, an approach for enabling the prediction of platform-level performance from infrastructure-level resource utilization was presented as a way to provide a cloud system with self-awareness capabilities that can enable in the future self-regulation techniques such as auto-scaling based on online intelligent techniques for early detection of service performance degradation and proactive decision making.

Using the methodology proposed in this project and after the evaluation of the performance achieved by different ML techniques for both KPIs prediction and SLA classification, RF-based techniques showed the highest performance for the tasks defined in this project. High accuracy given by a NMAE below 10.3% for the regressor and an accuracy score above 99.9% for the classifier, show the feasibility of the prediction models generated for service performance prediction.

Different challenges were faced during the development of this project: The definition of online KPIs that can enable early decision-making, the definition of SLA thresholds, the management of large set of features and class-imbalance, the need to have a fix number of features when having a dynamic number of cluster nodes, the inference of a good general model for all the defined experimental scenarios, the saturation of computational and networking resources, among others.

In this project, initial approaches were taken in order to tackle these challenges. However, some of the identified issues that can be further investigated are the following ones. Define the problem using forecasting instead of prediction and using ML techniques such as LSTM. Extend the work using Online Machine Learning techniques to handle data stream analysis using for example incremental learning with the definition of mini-batches and sliding-window for concept drift handling. Test the methodology proposed on data streaming processing framework such as Apache Spark. Compare the effect of using bare metal resource utilization metrics against container-based metrics collected from Docker stats. Compare the effect of using percentile metrics for aggregation instead of simple average. Test if the networking resources can be saturated when defining more reducers, including a study on how data locality affects the analysis. Evaluate the definition of KPI in terms of instant scaling factor, for example defining SLA violation if the service does not scale linearly when the input data size increases. Be more rigorous in the way the SLA thresholds are defined. Study the effect of failures and root-cause analysis. Study effects of heterogeneous environments. Evaluate using more representative benchmarks such as [47]. Either use nodes with lower specs or introduce VMs in the testbed. Integrate the developed model infrastructure-platform with a model mapping platform-software level by defining metrics such as execution time from the
end-user perspective. Study how sharing resources among several users would impact the models. Integrate the prediction/forecasting models with a cloud elasticity service.
Literature


[31] ——, “Predicting service metrics for cluster-based services using real-time analytics”, in *2015 11th International Conference on Network and Service Management (CNSM)*, November 2015, pp. 135–143.


Appendix

A Demonstrator Development

A demonstrator visualizing both Resource Utilization Space (RUS) as a 3D space and Carried Load Space (CLS) as a 2D space. For the selection of the features to be included in the Resource Utilization Space (RUS), an approach is taken based on domain knowledge combined with a feature relevance study generated from the Random Forest (RF) model when using the full dataset as shown in Fig. A.1

![Figure A.1: RUS Visualization based on Feature Relevance Analysis given by RF.](image)

The Carried Load Space (CLS) is defined by the feasibility region based on the samples collected during experimentation. This region is characterized by a high usage region and a low usage region. A possible use case may refer to the detection of high usage which is hold for a long enough time then the scaling tool offered by the infrastructure level spawns the service deployment in additional worker nodes. This scaling-out process allows executing more parallel tasks resulting in a higher throughput and the SLA fulfillment.

The demonstrator is developed refactoring the python code previously generated for the key-value store project developed in [3]. This tool was developed based on the following steps:

- Use triangulation to generate points in Resource Utilization Space (RUS) based on the min-max ranges from the collected data
- Train a simple batch regressor mapping Carried Load Space (CLS) to Resource Utilization Space (RUS) to generate points on Carried Load Space (CLS) and classify them based on the pre-defined thresholds

- Use these labels to classify Resource Utilization Space (RUS) from the previous step

- The RUS visualization uses hull delaunay and the Carried Load Space (CLS) visualization uses 2D convex hull

This demonstrator shows the following information:

- Resource and carried load spaces learned with batch learning

- Online prediction with learned models (Regression mapping RUS-CLS and Classification on CLS)

- Case by case study for each experiment using an interactive tool in Jupyter Notebook that allows to change the type of experiment to display

- Performance evaluation for both regressor and classifier (% Mean Absolute Error and Confusion Matrix)

Figs. A.2 and A.3 show how the Resource Utilization Space (RUS) and Carried Load Space (CLS) look like for the experimental scenarios when running the WC and TS benchmarks.
Figure A.2: RU and CL Spaces: WC.
Figure A.3: RU and CL Spaces: TS.