POLITECNICO DI MILANO

Dipartimento di Elettronica Informazione e Bioingegneria
Doctoral Program in Information Technology

Performance Models, Design and Run Time Management of Big Data Applications

Doctoral Dissertation by:
Eugenio Gianniti

Supervisor:
Prof. Danilo Ardagna
Co-Advisor:
Dott. Michele Ciavotta
Co-Advisor:
Dott. Marco Lattuada
Tutor:
Prof. Luciano Baresi
The Chair of the Doctoral Program:
Prof. Andrea Bonarini

2018 — XXX Cycle

Alla mia famiglia, sia qui che lassù

Abstract

Nowadays the big data paradigm is consolidating its central position in the industry, as well as in society at large. Lots of applications, across disparate domains, operate on huge amounts of data and offer great advantages both for business and research. As data intensive applications (DIAs) gain more and more importance over time, it is fundamental for developers and maintainers to have the support of tools that enhance their efforts since early design stages and until run time. The present dissertation takes this perspective and addresses some pivotal issues with a quantitative approach, particularly in terms of deadline guarantees to ensure quality of service (QoS).

Technically interesting scenarios, such as cloud deployments supporting a mix of heterogeneous applications, pose a series of challenges when it comes to predicting performance and exploiting this information for optimal design and management. Performance models, with their potential for what if analyses and informed design choices about DIAs, can be a major tool for both users and providers, yet they bring about a trade-off between accuracy and efficiency that may be tough to generally address. The picture is further complicated by the adoption of the cloud technology, which means that assessing operating costs in advance becomes harder, but also that the contention observed in data centers strongly affects big data applications' behavior. For all these reasons, ensuring QoS for novel DIAs is a difficult task that needs to be addressed in order to favor further development of the field.

Over this background, the present dissertation takes two main routes towards facing such challenges. At first we describe and discuss a number of performance models based on various formalisms and techniques. Among these, there are both basic models aimed at predicting specific metrics, like response time or throughput, and more specialized extensions that target the impact on big data systems of some design decisions, e.g., privacy preserving mechanisms or cloud pricing models. On top of this, the proposed models are variously positioned across the spectrum between efficiency and accuracy, thus enabling different trade-offs depending on the main requirements at hand. This is relevant in the second main part of this dissertation, where performance prediction is at the core of some formulations for capacity allocation and cluster management. In order to obtain optimal solutions to these problems, in one case at design time and in the other at run time, we adopt both mathematical programming and several performance models, according to the different constraints on solving times and accuracy.

More in detail, we propose performance models based on queueing networks (QNs), stochastic well formed nets (SWNs), and machine learning (ML). This variety is justified by the different uses of each methodology. ML pro-

vides algebraic formulas for execution times, which are perfectly fit to be added as constraints in our optimization problems' mathematical programming formulations, thus yielding initial solutions in closed form. Since ML can reliably provide accurate predictions only in regions properly explored during the training phase, the optimal solution is searched via a simulation-optimization procedure based on analytical models like QNs or SWNs, which in contrast are quite insensitive to the parameter range of evaluation, being devised from first principles. These kind of models boast relative errors below 10 % on average when predicting response times.

In terms of optimization, first of all we consider the design time problem of capacity allocation in a cloud environment. The design space is explored via both ML and simulation techniques, so as to choose the best virtual machine type in the catalog offered by cloud providers and, subsequently, determine the minimum cost configuration that satisfies QoS constraints. We show also how this optimization approach was applied during the design phase of a tax fraud detection product developed by industrial partners, i.e., NETF Big Blu. Afterwards we also considered the run time issue of finding the minimum tardiness schedule for a set of jobs when the current workload exceeds predictions and the deployed capacity is not enough to ensure the agreed upon QoS. Thanks to the varied efficiency of performance models, it is possible to solve the design time problem in a matter of hours, whilst run time instances are solved within minutes, consistently with the different requirements.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Many analysts point out that during these years technologies and methodologies that fall within the sphere of big data have swiftly pervaded and revolutionized many sectors of industry and economy, becoming one of the primary facilitators of competitiveness and innovation [65].

IDC reports that big data used to concern highly experimental projects, yet its market is growing from $130 billion in 2016 to $203 billion in 2020, with a compound annual growth rate of 11.9 %, with the banking and manufacturing industries leading in terms of investment [131]. Big data applications offer many business opportunities that stretch across industries, especially to enhance performance, as in the case of recommendation systems. Furthermore, data intensive applications (DIAs) can also help governments in obtaining accurate predictions, for instance quality weather forecasts to prevent natural disasters and ease the development of appropriate policies to improve the population's life quality. To corroborate these considerations, notice that the Obama government announced $200-million worth of investment to boost big data related industries and positioned this strategy into the national agenda in 2012. In addition, big data systems are increasingly exerting a central force on society, thus requiring the development of intelligent systems providing quality of service (QoS) guarantees to their users.

This dissertation reckons DIAs' importance in today's economy and tackles some relevant problems linked to their adoption. Any fruitful approach to the optimization of big data applications must rely on effective performance models, which are basic tools needed for the prediction of execution times or the determination of costs given QoS constraints. For this reason, the initial part of this dissertation discusses and compares a range of performance models, based on various formalisms and techniques. Building on top of these, we also propose novel solutions to optimization problems that play a major role in DIAs' design and operation. In particular, we will detail formulations for the capacity planning problem at design time, as well as for the run time management of workload peaks.

This chapter proceeds as follows. Section 1.1 presents the main challenges to face when dealing with DIAs in the cloud. Later on, Section 1.2 introduces

the most relevant big data frameworks, which were studied in the development of this dissertation, then Section 1.3 similarly introduces convolutional neural networks. After that, Section 1.4 motivates the choice to investigate performance models and summarizes the proposed ones, while Section 1.5 is about the optimization techniques. Section 1.6 acknowledges the international research projects that supported this work and Section 1.7 introduces the main questions that drive the content of this dissertation. The chapter ends with a list of the relevant publications where I contributed and the thesis organization.

1.1 Challenges for QoS Aware Big Data Systems

As DIAs acquire a central position in society, the IT field should shift from simply building systems to developing intelligent systems that provide QoS. Yet, predicting the performance of big data applications in scenarios of technical interest, notably a mix of applications running concurrently in a cloud system, is very challenging. Big data applications are characterized by changing behavior during execution: for instance, they require initially a lot of CPUs, then a lot of network capacity, to later switch between the two, with sometimes complex patterns. Moreover, to cope with the large amount of data, such applications often run in parallel stages. The performance (and thus QoS) of parallel applications is often harder to predict due to synchronization overheads. In summary, the most relevant challenges are:

Performance prediction via models Designing new models to predict the performance of applications, in terms of execution time, given certain resources, is key to providing QoS to application customers. Such models should be accurate and efficient, i.e., provide an estimate quickly. The use of accurate models is beneficial for both cloud providers and end users: for cloud providers, models can trigger run time adaptations to provide QoS guarantees; for end users, they can support what if analysis and enable taking more informed decisions on the resources to use. It is important that the models run reasonably fast, i.e., provide responses quick enough to drive run time adaptations. However, model accuracy and efficiency are two often conflicting objectives. More sophisticated models, capturing in more details different aspects of the application execution, are often more accurate, but also very costly to run. Thus finding the best trade-off between these two goals is a major challenge.

Cost predictability in the cloud Designing new models to estimate the costs in terms of cloud resources to run big data applications is key. An accurate estimate enables more efficient scheduling of resources, including cost-effective utilization of data centers.

Ensuring QoS on a budget Big data applications are supported by cloud infrastructures that, for resource contention, can be affected by performance decline. For this reason, one of the major challenges for big data applications is to define mechanisms and policies that implement resource partitioning and management in a way that cloud data centers' resources are used efficiently, providing differentiated service levels to customers according to the price of the resources.

In relation to the above challenges, at first this dissertation proposes various performance models and compares the advantages each brings to the table. Undoubtedly these models give a strong contribution for the prediction of response times and other performance metrics, but they can also be used to assess DIAs' resource requirements. Moreover, they are at the core of the optimization methods presented as the other important part of this dissertation. The choice among a gamut of alternative modeling techniques is instrumental to face the different issues that arise in the diverse scenarios of interest. Specifically, the adoption of optimization at design or run time is characterized by different constraints, for instance in terms of the time taken to obtain a solution or the required accuracy.

1.2 Big Data Frameworks

One of the pillars on which the big data revolution is based is the MapReduce paradigm, which has allowed for massive scale parallel analytics [79]. MapReduce, a programming model and a scalable and fault tolerant run time environment [36], is the core of Apache Hadoop, open source framework that has proven capable of managing large datasets over either commodity clusters or high performance distributed topologies [133].

The MapReduce framework became the most popular platform for data analytics because of its simplicity, generality, and maturity [141]. A data processing request under the MapReduce framework, called a job, consists of two types of tasks: map and reduce. A map task reads one data chunk and processes it to produce intermediate results, then reduce tasks fetch the partially processed data and carry out further computation to generate the final result [129].

Hadoop is an open source implementation of MapReduce ready for production deployments and used for applications like log file analysis, database (DB) querying, web indexing, report generation, machine learning research, scientific simulation, bioinformatics, and financial analysis [59, 132]. Hadoop's success has been planetary; it attracted the attention of both academia and industry as it overtook the scalability limits of traditional data warehouse and business intelligence solutions [79]. For the first time, processing unprecedented amounts of structured and unstructured data was within reach, thus opening up, suddenly, a whole world of opportunities.

From the technological perspective, MapReduce is capable of analyzing very efficiently large amounts of unstructured data, i.e., it is a viable solution to support both the variety and volume requirements of big data analyses [77]. Cloud platforms make MapReduce an attractive framework for organizations that need to process large datasets, but lack the computing and human resources to install and manage a cluster. Moreover, Hadoop 2.x recently introduced a wide set of performance enhancements, such as SSD support, caching, and I/O barriers mitigation. IDC had estimated that Hadoop touched half of the world data by 2015 [71], supporting both traditional batch and interactive data analysis applications [112]. Paradoxically, the MapReduce paradigm, which has contributed so much to Hadoop's rise, is steadily declining in favor of solutions based on more generic and flexible processing models. Among these, Apache Spark is a framework that is enjoying considerable success and

that, according to analysts, is expected to dominate the market for the next decade [41].

The rigid division between map and reduce requires to subdivide a complex application into a directed acyclic graph (DAG) of MapReduce jobs, comprising tasks that perform a specific computation on partitions/splits of the input data. In this case, the MapReduce paradigm forces to store the results of each intermediate phase on disk, thus being unsuitable for applications requiring a low latency between different phases, along with general application QoS guarantees. Other frameworks, such as Tez [109] and Spark [138], have been introduced to address this problem. Although Tez can handle general DAGs of MapReduce phases, it still requires to write each stage's results on disk. On the other hand, Spark can exploit a set of primitives to request the caching of partial results in memory, thus allowing lower latency and better performance. Spark has been developed on the resilient distributed dataset (RDD) concept [137], a novel distributed memory abstraction providing a restricted form of memory sharing. In practice, Spark can easily obtain a 10x speedup over Hadoop on specific scenarios [138]. This motivates Spark's widespread adoption, which made it the leading framework for data science at scale. Acknowledging its newly acquired importance for big data, the focus of this dissertation shifted from MapReduce, which in this fast paced field can now be considered legacy, to Apache Spark.

1.3 Deep Learning Frameworks

Among DIAs, an important role is played also by neural networks (NNs). Nowadays, convolutional neural networks (CNNs) find application across industries, most notably for image recognition and classification tasks, which represented the first successful adoption of the technique [74]. Ranging from medical diagnosis to public security, deep learning (DL) methods are fruitfully exploited in a wide gamut of products. In addition to the established applications, there is ongoing work on the technique's adaptation for other use cases, like speech recognition [110] and machine translation [17]. Over time, many frameworks have been developed to provide high level APIs for CNN design, learning, and deployment. Among the most well known, we recall Torch,[1] PyTorch,[2] TensorFlow,[3] and Caffe.[4]

Contrasting to big data frameworks, CNNs generally do not process unstructured data, instead networks themselves are somewhat tailored for the intended input data. Adopting the image recognition example to support intuition, CNNs are peculiar in that they consist of two main portions: an ordinary NN acting as classifier, thus distinguishing different image categories, constitutes the final end of the structure, while a convolutional part automatically extracts features to feed into the classifier. In layman's terms, the learning process enables the convolutional layers to recognize high level features such as beaks or paws, which in turn help the classifier in telling cats from birds.

[1] http://torch.ch
[2] http://pytorch.org
[3] https://www.tensorflow.org
[4] http://caffe.berkeleyvision.org

Usually DL models are trained relying on GPGPU systems (even in clusters for experimental environments [130]), which allow to achieve from 5 up to 40x time improvement when compared to CPU deployments [18]. Motivated by the relevance assumed by these applications, in the following we also propose two performance models specifically tailored for CNNs.

1.4 Performance Modeling for DIAs

In spite of all the fuss around big data technologies, it is still undeniably true that fully embracing them is a very complex process. Many efforts have been made to make this technology accessible, but establishing a production ready deployment is time consuming, expensive, and resource intensive. Not to mention the fact that fine tuning is still often perceived as a kind of occult art.

It is widely held that there is a clear need for an *easy button* to accelerate the adoption of big data analytics [55]. That is why many companies have started offering cloud-based big data solutions, like Microsoft HDInsight, Amazon Elastic MapReduce, or Google Cloud Dataproc, while IDC estimates that, by 2020, nearly 40 % of big data analyses will be supported by public clouds [49]. The advantages of this approach are manifold. For instance, it provides an effective and cheap solution for storing huge amounts of data, whereas the pay per use business model allows to cut upfront expenses and reduce cluster management costs. Moreover, the elasticity can be exploited to tailor clusters capable to support DIAs in a cost-efficient fashion. Yet, provisioning workloads in a public cloud environment entails several challenges. In particular, the space of configurations (in particular, in terms of nodes type and number) is very large, thus identifying the exact cluster configuration is a complex task, especially in light of the consideration that the blend of job classes in a specific workload and their resource requirements may also vary over time.

At the very beginning, MapReduce jobs were meant to run on dedicated clusters to support batch analyses via a FIFO scheduler [103, 105]. Nevertheless, DIAs have evolved and nowadays large queries, submitted by different users, need to be performed on shared clusters, possibly with some guarantees on their execution time [142, 143]. This is not a loose requirement, indeed, as one of the major challenges [83, 122] is to predict the application execution times with a sufficient degree of accuracy. In such systems, capacity allocation becomes one of the most important aspects. Determining the optimal number of nodes in a cluster shared among multiple users performing heterogeneous tasks is a relevant and difficult problem [122].

Unfortunately, modeling the performance of such systems is very challenging. Indeed, production Hadoop environments are nowadays very large massively parallel systems where map and reduce tasks coordinate exhibiting precedence constraints and strict synchronization barriers. Additionally, in our context, the stakeholders interested in the performance evaluation of Hadoop processes are its users rather than its developers. Therefore, the complexity and novelty of these systems together with the lack of full knowledge of their development details make unclear the concepts that should be included in a performance model in order for them to be both accurate and manageable by performance evaluation tools.

Moreover, with Hadoop 2, resources are dynamically allocated between

the map and reduce stages. While in early Hadoop versions CPU *slots* and other resources were separated between mappers and reducers using a static approach, in Hadoop 2 *containers* (both for MapReduce and Spark) are distributed among ready tasks in a dynamic fashion by YARN. On the one hand, this allows a better cluster utilization, on the other hand performance modeling became much more difficult. In Spark, cluster resources are scheduled to process part of the operations on RDDs: to obtain an RDD, Spark first builds a DAG with its dependencies, then processes each stage providing a certain amount of resources, based on data locality.

Because of all these reasons, predicting the execution time of Hadoop or Spark jobs is usually done empirically through experimentation, requiring a costly setup [54]. Performance prediction models are extremely useful to aid development and deployment of big data applications, either for design time decisions or run time system reconfiguration. Design time models can help, e.g., to determine the appropriate size of a cluster or to predict the budget required to run Hadoop or Spark in public clouds. Such models can be used also at run time, allowing for a dynamic adjustment of the system configuration [11, 104], e.g., to cope with workload fluctuations or to reduce energy costs.

Analytically modeling DIAs is very challenging due to the great number of parameters that have to be investigated. To ensure analytical model (AM) tractability in such complex systems, AM-based performance models typically rely on simplifying assumptions that reduce their accuracy. On the other hand, machine learning (ML) deals with the study and construction of algorithms that can learn from data and make predictions on it without a priori knowledge about the internals of the target system. In recent years, a growing number of successful researches were done to explore the possibility of using ML techniques to predict the performance of complex computer systems [62, 76, 135]. To be able to predict accurately, the ML model should be built during a training phase with a sufficient amount of experimental data from different workloads, using various parameters and configurations. However, running several experiments in a cloud environment would be costly and time consuming. On top of this, though ML often provides good accuracy in regions for which it is well trained, it shows poor precision in regions for which none or very few samples are known.

Gray box modeling [42, 44, 63] is a new approach for performance modeling and prediction that tries to achieve the best of the AM and ML worlds by mixing the two. Such models can be exploited to support design time decision making during the development and deployment phases of big data applications. These models can then be also kept alive at run time to conduct the dynamic adjustment of the system configuration [104]. In this context, the performance models proposed in the present dissertation span the whole spectrum from AMs to MLs, also with a hybrid approach aimed at attaining similar accuracy despite the use of less operational data, which entails savings on ad hoc experimental deployments.

In spite of the widespread adoption of DL systems, still there are few studies taking a system perspective that aim at investigating how, for example, the training time changes when running on different GPGPUs or by varying the number of training iterations or the batch size [18, 58]. DL applications are characterized by a large number of design choices that often do not ap-

ply readily to other domains or hardware configurations, up to the point that even advanced users with considerable DL expertise fail at identifying optimal configuration settings [58].

1.4.1 Contributions

The originality of this dissertation consists in an array of modeling techniques capable of catching the system behavior under the dynamic assignment of the available cluster resources. We assume that the cluster is governed by the Capacity Scheduler, which partitions the available resources among multiple customers through queues, each queue being regulated by a FIFO policy. Based on this assumption, we devised several performance models relying on different formalisms, so as to evaluate their relative accuracy and efficiency and tailor them to the specific requirements of each use case. In particular, we investigated queueing networks (QNs), stochastic well formed nets (SWNs), fluid Petri nets, discrete event simulators (DESs), as well as ML approaches.

When the focus is on model expressiveness rather than fast prediction, for instance during the design phase, it is possible to estimate DIAs execution times for multiple users and under unreliable resources. In particular, we analyze a cloud-based scenario where the cluster, to save execution costs, includes also spot virtual machines (VMs) [48]. The utilization of spot VMs offers large discounts in VM prices, with the drawback of a non-guaranteed availability level. We combine the performance and availability dynamics of cloud resources in a single *performability* model that allows for evaluating how failures caused by a sudden deallocation of VMs by the cloud providers degrade system performance.

In this dissertation, we also present a method to learn performance models for CNNs running on a single GPGPU. The main metrics under investigation are the forward time, relevant to quantify the time taken for classification when the trained network is deployed, and the gradient computation time, which on the other hand is important during the learning phase.

The validation of all the proposed models has been carried out with experiments on real systems, considering a number of target deployments. Namely, we considered public clouds with Amazon EC2 and Microsoft Azure HDInsight, community clouds with CINECA, the Italian supercomputing center, as well as on premises deployments with an internal installation, based on IBM POWER8 processors, at Politecnico di Milano. In order to make the validation both reproducible and reliable, we chose the TPC-DS benchmark, which is an industry standard for data warehouse and business intelligence applications. Alongside these analyses based on the benchmark, we also considered some case studies proposed by industrial partners, including also ML workloads. The presented simulation models show, on average, a good accuracy with respect to measurements: their mean relative errors are 14.13 % for QNs and 9.08 % for SWNs.

1.5 Capacity Planning and Resource Management Optimization

Given a set of performance modeling techniques, with their pros and cons, the next step of this dissertation is the development of optimization methods to solve problems related to the management of DIAs in the cloud. Specifically, we focused on the issues of capacity planning at design time, while at run time on the rebalancing of the available resources to meet the requirements of newly submitted jobs, possibly going beyond the foreseen workload and associated capacity.

We formulate the design time capacity planning problem by means of a mathematical model, with the aim of minimizing the cost of cloud resources. The problem considers multiple VM types as candidates to support the execution of big data applications from multiple user classes. Cloud providers offer VMs of different capacity and cost. Given the complexity of virtualized systems and the multiple bottleneck switches that occur in executing DIAs, very often the largest available VM is not the best choice from either the performance or performance/cost ratio perspective [54, 142]. Through a search space exploration, our approach seeks the optimal VM type and number of nodes considering also specific cloud provider pricing models (namely, reserved, on demand, and spot instances). The underlying optimization problem is NP-hard and is tackled by a simulation-optimization procedure able to determine an optimized configuration for a cluster managed by the YARN Capacity Scheduler. DIA execution times are estimated by relying on a gamut of models, including ML and simulation based on QNs, stochastic Petri nets (SPNs) [9], as well as an ad hoc simulator, dagSim [3], especially designed for the analysis of applications involving a number of stages linked by DAGs of precedence constraints. This property is common to legacy MapReduce jobs, workloads based on Apache Tez, and Spark-based applications.

Analogously, the run time problem of resource reallocation is formulated as a distinct mathematical programming model whose objective is the minimization of tardiness. In this case the focus shifts to private clouds, where the previously applied pricing model is not relevant. Along the same lines, it is not possible to choose a different VM type, since this decision was already taken during the design phase. Exploiting another simheuristic procedure, it is possible to obtain the optimal reallocation of resources that enables hard deadline DIAs to meet their service level agreements (SLAs) and loosens the constraints on soft deadline jobs, in order for them to achieve the minimum overall weighted tardiness.

1.6 Acknowledgment

This dissertation has been developed within the framework of two H2020 projects: DICE and EUBra-BIGSEA. The former has as main goal a DevOps framework for designing DIAs and exploiting information extracted from their deployments to improve on such design, whilst the latter aims at a run time environment to provide QoS guarantees for big data, both in data centers and in the cloud.

Relying on the hereby presented performance models and optimization

methods, we obtained two main outcomes: D-SPACE4Cloud[5] and several modules in the EUBra-BIGSEA ecosystem. D-SPACE4Cloud is a piece of software designed to help system administrators and operators in the capacity planning of shared big data clusters hosted in the cloud, so as to support both batch and interactive applications with deadline guarantees. We believe that being able to successfully address this problem at design time enables developers and operators to make informed decisions about the technology to use, while also allowing for the full exploitation of the potential offered by the cloud infrastructure. On the other hand, at run time it is no more possible to take far reaching choices, yet the EUBra-BIGSEA architecture enables the optimal management of the available resources, so as to reduce the impact on QoS of unforeseen workload spikes.

1.7 Research Questions

Overall, this dissertation's main contributions revolve around four research questions. These range from the accuracy of performance models and prediction techniques, but also their time efficiency, to the effectiveness of optimization procedures, even when applied in time constrained scenarios. It is also relevant to investigate the impact of different deployment options on performance, with particular attention to the savings enabled by non-obvious interactions among workloads and underlying computational capabilities. In the end, the proposed optimization techniques may allow for design choices that go beyond simple cluster sizing, then such a possibility should be assessed. The following paragraphs expand with more details the above mentioned problematics.

Research question 1. *Which performance models can be applied to DIAs in an accurate and efficient way? In particular, which have fitting characteristics for design time optimization techniques or, alternatively, for run time management of big data deployments?*

Searching for the optimal configurations to deploy DIAs implies the basic requirement of predicting with a fair confidence their performance, depending on the amount and type of allocated computational resources. If design time optimization is less constrained in terms of convergence times, conversely when operating at run time it is fundamental to shrink the time taken for the optimization procedure, so as to keep at a minimum the impact on DIAs execution. Hence, it is relevant to investigate the trade-offs between accuracy and prediction speed enabled by different alternative techniques, thus determining which better fit either application scenario and highlighting the compromises that might possibly be needed.

Research question 2. *How to identify the minimum cost configuration at design time and how to manage it at run time under high load conditions? Are the proposed optimization methods accurate?*

One of the major issues when dealing with complex pieces of software, such as DIAs, is that they provide lots of configuration parameters, a number of which have effects on the overall performance. This problem is further

[5]DICE System Performance and Cost Evaluation for Cloud

worsened by the vast catalog of alternative deployment choices enabled by the cloud. In a similar setting, it is fundamental to devise appropriate methods capable of efficiently exploring the state space, since an exhaustive search would be utterly impossible. Even more so when the goal is the run time management of DIAs clusters, as the more stringent optimization time constraints exacerbate the issue. In the end, applicability of the proposed techniques heavily depends on their accuracy, hence it is important to assess it via an extensive experimental campaign.

Research question 3. *When looking for the minimum cost deployment, are there any dominant configurations? Is it possible, instead, that different workloads lead to different minimum cost configurations? What is the impact of providers' catalogs on these considerations?*

Common practices tend to associate specific classes of instances to applications based on the matching among VMs' computational capabilities and DIAs' requirements. An immediate example is offered by Apache Spark, which requires a large central memory on each worker node, so as to achieve faster processing in iterative applications via the caching of RDDs. Cloud providers often offer computing and memory optimized instances to satisfy such needs. The focus in this research question is on investigating whether such a preliminary choice is always optimal, or if the complex dependencies between frameworks and deployment options enable different minimum cost configurations based on varying concurrency levels, SLAs, and so forth. Moreover, it is interesting to assess whether such behavior can be observed also across various providers.

Research question 4. *Are these techniques useful to investigate application architectural choices at design time?*

Several architectural choices might have an impact on DIAs' observed performance and this, in turn, affects the optimal configuration for what concerns both the type and number of allocated resources. A natural implication is the adoption of the devised methods for modeling and optimization at design time, when it is possible to quantitatively explore several choices and to assess their effect on the final performance and operational costs.

Personal Publications

[8] Danilo Ardagna, Enrico Barbierato, Athanasia Evangelinou, Eugenio Gianniti, Marco Gribaudo, Túlio B. M. Pinto, Anna Guimarães, Ana Paula Couto da Silva, and Jussara M. Almeida. "Performance Prediction of Cloud-Based Big Data Applications." In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE '18. Berlin, Germany: ACM, 2018, pp. 192–199. ISBN: 978-1-4503-5095-2. DOI: 10.1145/3184407.3184420.

[9] Danilo Ardagna, Simona Bernardi, Eugenio Gianniti, Soroush Karimian Aliabadi, Diego Perez-Palacin, and José Ignacio Requeno. "Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets." In: *16th International Conference on Algorithms and Architectures for Parallel Processing*. Ed. by Jesús Carretero, Javier García Blas, Ryan K. L. Ko, Peter Mueller, and Koji Nakano. Vol. 10048. Lecture Notes in Computer Science. Springer, Dec. 2016, pp. 599–613. ISBN: 978-3-319-49582-8. DOI: 10.1007/978-3-319-49583-5_47.

[15] Ehsan Ataie, Eugenio Gianniti, Danilo Ardagna, and Ali Movaghar. "A Combined Analytical Modeling Machine Learning Approach for Performance Prediction of MapReduce Jobs in Cloud Environment." In: *SYNASC* (Timisoara, Romania). 2016.

[32] Michele Ciavotta, Eugenio Gianniti, and Danilo Ardagna. "Capacity Allocation for Big Data Applications in the Cloud." In: *ICPE'17 Companion* (L'Aquila, Italy). Apr. 2017, pp. 175–176.

[33] Michele Ciavotta, Eugenio Gianniti, and Danilo Ardagna. "D-SPACE4Cloud: A Design Tool for Big Data Applications." In: *ICA3PP* (Granada, Spain). Lecture Notes in Computer Science 10048. Springer, Dec. 2016, pp. 614–629.

[51] Eugenio Gianniti, Danilo Ardagna, Michele Ciavotta, and Mauro Passacantando. "A Game-Theoretic Approach for Runtime Capacity Allocation in MapReduce." In: *CCGrid* (Madrid, Spain). May 2017, pp. 1080–1089.

[52] Eugenio Gianniti, Alessandro Maria Rizzi, Enrico Barbierato, Marco Gribaudo, and Danilo Ardagna. "Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications." In: *ACM Performance Evaluation Review* 44.4 (Mar. 2017), pp. 23–36. DOI: 10.1145/3092819.3092824.

[53] Eugenio Gianniti, Alessandro Maria Rizzi, Enrico Barbierato, Marco Gribaudo, and Danilo Ardagna. "Fluid Petri Nets for the Performance Evaluation of MapReduce Applications." In: *InfQ* (Taormina, Italy). Oct. 2016.

[70] Safia Kalwar, Eugenio Gianniti, Joas Yannick Kinouani, Youssef Ridene, and Danilo Ardagna. "Performance Degradation and Cost Impact Evaluation of Privacy Preserving Mechanisms in Big Data Systems." In: *New Frontiers in Quantitative Methods in Informatics*. Ed. by Simonetta Balsamo, Andrea Marin, and Enrico Vicario. Vol. 825. Communications in Computer and Information Science. Springer, 2017, pp. 82–96. ISBN: 978-3-319-91631-6. DOI: 10.1007/978-3-319-91632-3_7.

[89] Marzieh Malekimajd, Danilo Ardagna, Michele Ciavotta, Eugenio Gianniti, Mauro Passacantando, and Alessandro Maria Rizzi. "An Optimization Framework for the Capacity Allocation and Admission Control of MapReduce Jobs in Cloud Systems." In: *The Journal of Supercomputing* (May 2018). ISSN: 1573-0484. DOI: 10.1007/s11227-018-2426-2.

Manuscript Organization

This dissertation is organized as follows. To begin with, Chapter 2 outlines related work and the state of the art. Then Chapter 3 provides details about the performance models developed for this research, thus also addressing research question 1. Chapter 4 describes the problem setting and the formulations proposed for optimization, both at design and run time, which entails research question 2. Further on, Chapters 5 and 6 discuss experimental results that validate both performance models and optimization techniques. Alongside validating research questions 1 and 2, they also show results and case studies about research questions 3 and 4. In the end, Chapter 7 draws the conclusions of this dissertation and hints at possible future work.

CHAPTER 2

State of the Art

This chapter presents the basics of the relevant technologies and frameworks, as well as various literature proposals relevant for this work. First of all, Section 2.1 describes quite extensively both Apache Hadoop and Apache Spark, which were the main frameworks subject to investigation for modeling and optimization. Then Section 2.2 discusses several formalisms and their application to performance modeling of DIAs, while Section 2.3 is about examples of the use of hybrid ML techniques. Afterwards, Section 2.4 shows related work relevant to the aspects of capacity planning and resources management for big data. In the end, Section 2.5 underlines open issues in the current literature and mentions how this dissertation contributes.

2.1 Overview of Technologies and Frameworks

Nowadays, Apache Hadoop 2 is the most widespread solution for handling massive dataset on clusters of commodity hardware, while Spark is the most promising framework that will probably support the execution of big data applications for the next 5–10 years [41].

2.1.1 MapReduce and Hadoop

MapReduce is a general algorithm composed of two main functions, map and reduce. The name was given by Google [37] through its first commercial implementation, which allows for handling huge datasets in a fault-tolerant, distributed framework. The most widespread open source implementation of this programming paradigm is Apache Hadoop.[1]

Hadoop in the root uses Apache Lucene, which is the open source API for information retrieval, specifically one of its sub-projects: the web search engine Apache Nutch, created as webcrawler in 2002. Google in 2003 published the architecture of its distributed file system, Google File System, and in 2004 a paper explaining the MapReduce paradigm [37], providing in such a way

[1] http://hadoop.apache.org

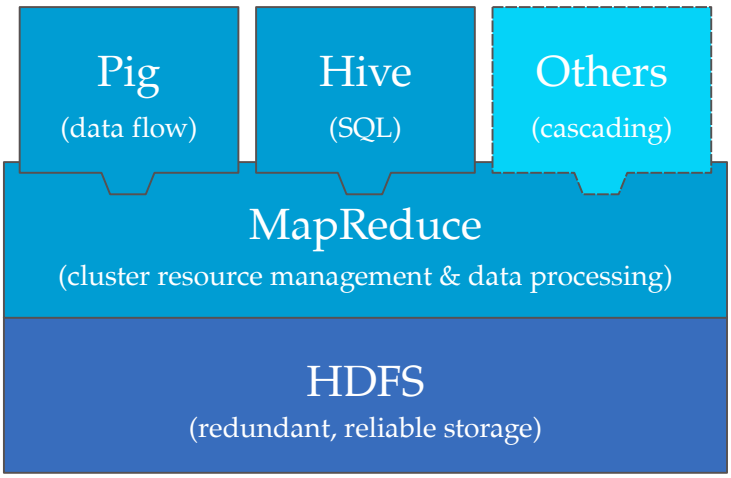


Figure 2.1 – Hadoop v1 architecture

two important technologies necessary to overcome the scalability limits of the Nutch project. In 2006 Hadoop became an independent sub-project of Lucene and a top level Apache project in 2008.

The first version of Hadoop had only a framework for the MapReduce programming model. Every application that could be rewritten in terms of map and reduce phases could take advantage of Hadoop's distributed computation capabilities. Figure 2.1 shows how Hadoop uses the Hadoop Distributed File System (HDFS) and its related service as a storage layer, while the MapReduce framework enables users to submit their own applications as MapReduce jobs. Applications like Hive and Pig have been developed on top of Hadoop and, by hiding the underlying MapReduce engine, they provide the capability of using high level query languages to operate on data.

2.1.1.1 HDFS

The Hadoop Distributed File System was built as the infrastructure for the Nutch Project. It has its origin in the Nutch Distributed File System, which was specifically developed to overcome the lack of a distributed file system to handle big sized files with a cluster of barebone commodity hardware. This kind of file system had already most of the features and concepts that are found in HDFS. The lack of a user permission system, absence of quotas, and a much shorter set of configurable settings are, however, an exception.

In a Hadoop cluster, a master node running the NameNode service for HDFS is responsible for maintaining a file system tree with the location and properties of files in the so called FsImage file and metadata changes into a transaction log, the EditLog. Both the FsImage and the EditLog are stored in the local OS file system. Another relevant process in the HDFS infrastructure is the DataNode, running on every node and physically storing data, as can be seen in Figure 2.2. This last process is active on every node of HDFS and handles the requests coming from the NameNode about physical operations on files.

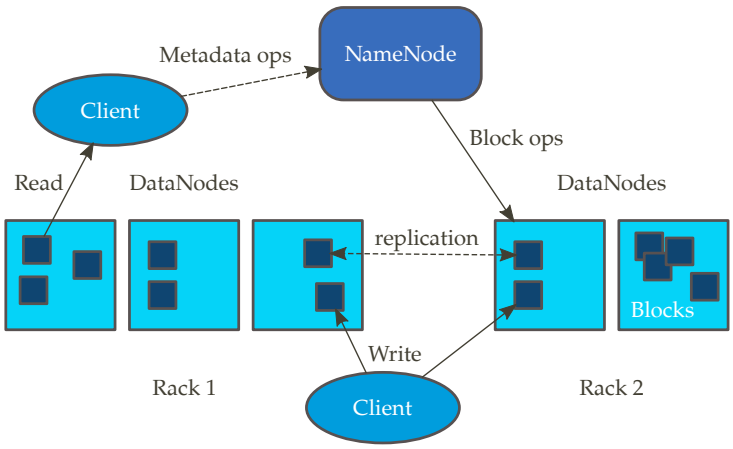


Figure 2.2 – HDFS architecture

Exploiting HDFS, MapReduce can operate over huge-sized files, typically at tera- or petabyte scale, by organizing them in blocks of fixed size and replicating them among different nodes. HDFS has a default replication factor of three. In order to provide data availability and reliability even upon different degrees of system failure, while reducing network bandwidth utilization, two replicas are, possibly, kept on different nodes of the same rack, and one replica on another rack. The data copying process takes place in a pipelined fashion. Since broadcasting the file from a unique source to the destination nodes would impose an excessive toll on local network resources, when the first DataNode starts receiving and writing data from the client, it forwards it to the second DataNode, and so does this, until all the DataNodes have their own replica. In this way the client has to forward data just once, spreading the workload farther. In case of node failures, the DataNodes communicate among themselves, without the intervention ofthe central NameNode, to keep the file system in a coherent state, as shown in Figure 2.2.

HDFS has been specifically developed to store files of huge size and provide a high throughput. For this reason data is expected not to change, maintaining a write-once-read-many access model. This is especially important if the relevant amount of network traffic that would be generated by a change in data content, caused by the update of every replica of every block where the modification occurred, is considered.

Client applications cannot directly write a file into HDFS: data has to be written first in a temporary file on the OS file system, then, once the file reaches the size of a HDFS data block, a request is sent to the NameNode, which will record the change in the file system structure and send back the location of the DataNode where the new data will be physically stored. Only upon file closure the NameNode will commit the file creation operation to the persistent log. Users can interact with the file system at the NameNode with a pseudo-POSIX interface, hiding the underlying NameNodes and DataNodes structure.

In the first version of HDFS, the NameNode was a single point of failure. In order to provide high availability, in recent versions two redundant Na-

meNodes are instantiated on different machines of the same cluster. At any point in time, one is in an active state, while the other is kept in a hot standby state to provide automated failover in case of necessity. The standby node can keep its state synchronized with the active one in two ways: either by communicating the changes through a shared common storage device, or via a group of separate daemons, called Journal Nodes, on which the active node can communicate the applied changes and from which the passive one can read them.

In case of user errors or situations when disaster recovery is needed, HDFS now provides snapshots of the file system. Snapshots are instantaneous and they record only the block list and the file size of a sub-tree of the file system, or even the entire file system, yet without replicating the actual data. Upon accidental deletion of a file, the related blocks are "protected" by the snapshot and so only its metadata is deleted. In this way, by restoring the snapshot it is possible to recover the previous state of the file system for that directory.

Another feature introduced with HDFS 2 is HDFS Federation. In the previous version of the file system, a single NameNode and a single namespace were allowed, now this limit is overcome with multiple and independent federated NameNodes and namespaces. In this way the horizontal scalability of the storage is supported by an horizontal scalability of the namespace, in addition to isolation and a throughput improvement.

2.1.1.2 MapReduce Applications

A MapReduce job, or application, includes input data, typically stored in HDFS), and user code defining the logic of map and reduce operations. The job execution can be seen as split into the map, shuffle, and reduce phases.

In the *map phase* input data is divided into appropriate splits, each fetched by a map task, making the number of map tasks data dependent. MapReduce takes advantage of data locality, so map tasks are assigned to the nodes that maximize the proximity of their needed data. Assigning a task on or close to the machine that stores its input data on local disks is essential in order to reduce network traffic and avoid congestion. For performance reasons, intermediate map output data is saved in the local file system of the worker, since replication would be time consuming and keep the network busy, whilst redundancy is not required for these partial values.

In the *shuffle phase* key-value pairs from all the mappers are fetched in parallel by some dedicated threads at the reducers, then the obtained files are sorted. This phase begins as soon as the first map task finishes, making the first data available for the reducers. The number of reducers is not data dependent, but can be manually set. Moreover, the set of keys is hash partitioned so that each key can be fetched by one and only one reduce task.

The *reduce phase* is the execution of the reduce logic on intermediate data. The persistent output is written to an output file system, typically HDFS.

It is important to notice that, since reducers do not focus on a specific source to collect their key-value pairs, data locality has less importance here, while a map task is instead executed as close as possible to the chunk of data it is assigned to. Since bandwidth usage optimization is a crucial element of MapReduce jobs performance, it is convenient to limit the amount of data transferred from the map tasks to a single reducer. By means of a so called

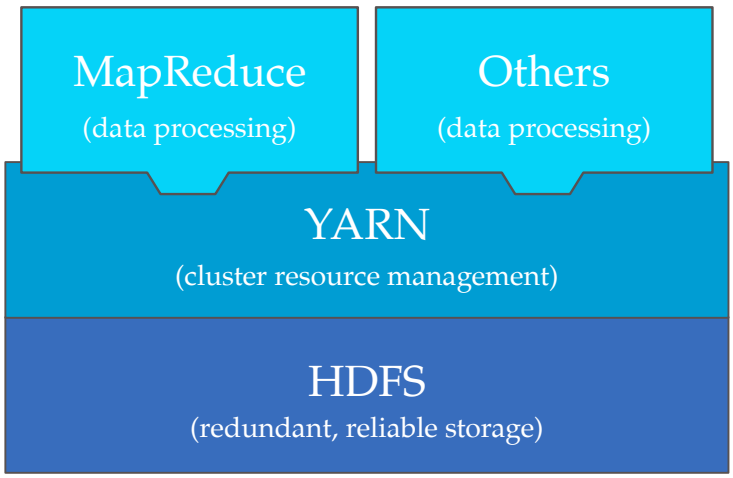


Figure 2.3 – Hadoop v2 architecture

combiner function, it is often possible to execute an aggregation of the output data of a map task on the physical node where it was executed, still keeping them in the form of a valid input for the subsequent reduce task.

2.1.1.3 Hadoop YARN

Hadoop YARN is the result of a complete overhaul of the framework performed in the development of the second version, yet it maintains backward compatibility with legacy MapReduce applications. In the first version's implementation, the whole cluster was managed by a single process, the JobTracker. Currently the two main JobTracker's functionalities, resource management and job scheduling/monitoring, have been split into separate daemons. The effects of this radical change on the architecture are a framework decoupled from the MapReduce paradigm and the movement of the application layer management away from the system daemons.

While initially Hadoop had the MapReduce engine as the only framework available to developers, now this layer is just one of the possible applications running on top ofYARN, as shown in Figure 2.3. For example, YARN allows users to execute Distributed-Shell applications on multiple worker nodes in the cluster. The system is still taking advantage of HDFS, but, as it leaves the storage layer, it faces important changes in the Hadoop architecture.

Since applications are not necessarily MapReduce jobs, it becomes too simplistic to organize the resources of nodes in terms of a fixed number of map and reduce slots. The cluster is now seen as a resource pool and requests are satisfied by assigning containers, providing a multiple of the fixed minimum amounts of memory, disk, network, and CPU resources to user code. In this way, applications can request and release resources according to their needs, gaining a high flexibility for the YARN resource model.

The JobTracker disappears and the resource management is now accomplished by a ResourceManager, which still resides on the master node and takes care of assigning containers to the requesting applications. Since the optimality of scheduling policies is strictly dependent on the kind of applica-

tions that are going to be run on the system, the ResourceManager works with a pluggable scheduler, allowing users to implement their own. By default, the Capacity scheduler can be replaced with the Fair scheduler or the old FIFO scheduler. This latter is convenient only in very small clusters with a limited workload, since its lack of job priority awareness and other advanced features makes it unsuitable for large shared clusters.

The other feature of the JobTracker, the job scheduling and monitoring, is now accomplished by ApplicationMasters. ApplicationMasters are per-application framework instances decoupled from the central system, typically operating on a slave node. When an application needs to start, the respective ApplicationMaster is the application-specific first component, which is deployed in a dedicated container. It then negotiates more containers with the ResourceManager on behalf of the single instances of that application, coordinates their execution, and monitors their resource consumption interacting with NodeManagers. This change is not trivial, since the scalability is no more limited by the capacity of the JobTracker and the sole role of the ResourceManager at the master node is just to schedule resources.

NodeManagers are per-node processes and can be seen as the evolution of TaskTrackers. They receive container requests from ApplicationMasters, run and possibly kill them, monitor the resources in use, manage logs and distributed caches, and periodically send heartbeats to the ResourceManager about the health and resource utilization of their nodes.

2.1.1.4 Hadoop 3

In September 2016, Apache introduced Hadoop 3, which was officially released in December 2017. The Apache community has incorporated many changes in Hadoop 3: some of the major new features follow.

Hadoop 3 uses erasure coding instead of replication. In this way, it is possible to sustain the same level of fault tolerance, despite consuming less storage space. According to the developers, this approach can lower storage costs up to two times.

Moreover, Hadoop 3 introduced support for GPUs. In the previous version, YARN could only handle memory and CPUs. Supporting GPUs enables large improvements for specific workloads, such as ML and DL.

Another relevant new feature is the support for cloud storage systems, for instance Amazon S3[2] and Microsoft Azure Data Lake.[3]

In the end, YARN now also supports Docker[4] containers. This addition enables mixing, on the same cluster, both big data and regular applications, while also offering package isolation. Thanks to the use of containerized applications, it is easy to work around compatibility issues among dependencies.

2.1.2 Spark

Apache Spark is a cluster computing platform designed to be fast and general purpose [139]. Given the fact that various use cases for Hadoop are somewhat

[2] https://aws.amazon.com/s3/

[3] https://azure.microsoft.com/en-us/solutions/data-lake/

[4] https://www.docker.com

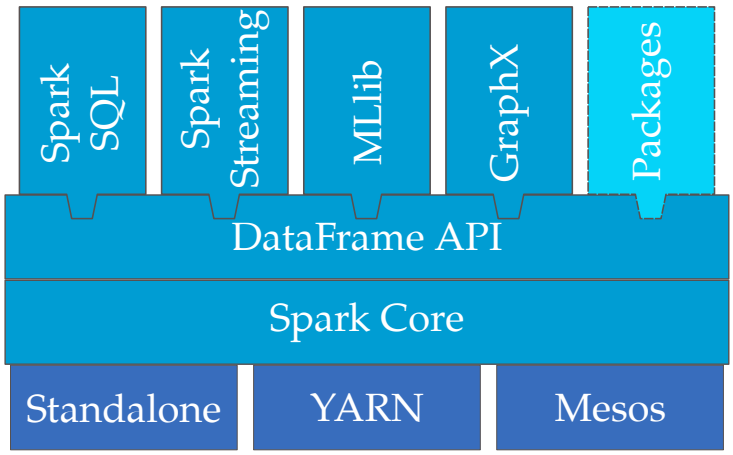


Figure 2.4 – Spark architecture

hindered by its continuous reliance on I/O to disk, often Spark is preferred due to its greater adoption of memory for intermediate results.

Speed is an important factor when the task is to process large datasets. The MapReduce model is extended in Spark so that it can efficiently support more types of computation, such as interactive queries and stream processing. Spark is also able to run computation in memory.

In Spark, workloads like iterative algorithms, interactive queries, and streaming do not require separate distributed systems, as was the case with previous big data tools. Spark supports these different workloads on the same engine, therefore it is easy and inexpensive to combine different processing types that are often needed in production data analysis pipelines. Integrating Spark with other big data tools is still possible: in particular, it is quite common to host Spark on Hadoop-managed clusters.

2.1.2.1 Spark Components

Apache Spark is responsible for scheduling, distributing, and monitoring applications at its core [139]. On top of its core engine, are available different higher level components for various tasks, such as real time streaming or ML. These components closely integrate with each other and users can easily combine them.

The Spark ecosystem is designed in a way that all higher level components benefit when the core engine adds an optimization. In addition, costs associated with deployment, maintenance, testing, and support are minimized thanks to the tight integration of the software stack, which avoids the need to rely on multiple independent frameworks.

Another advantage of Spark's ecosystem is the ability to build applications that combine different processing models. For example, data streaming and ML tasks can run simultaneously, possibly querying the same data in real time via SQL or accessing it through the Spark shell for ad hoc analyses.

Figure 2.4 shows the various building blocks of Spark architecture. In the following, each is presented with its main features and role in the big picture.

Spark is designed to easily scale up from one node to large clusters with many thousands of machines, hence the lowest layer in its architecture is occupied by several alternative cluster managers, such as YARN, Apache Mesos,[5] or a simple standalone scheduler distributed as part of the Spark project. User code can transparently run on any of these cluster management solutions, thanks to the decoupling provided by the core framework.

Spark Core is responsible for Spark's main functionality, such as task scheduling, memory management, fault recovery, and interacting with storage systems. RDDs, which are Spark's main programming abstraction, are defined in Spark Core. RDDs are immutable and partitioned collection of records distributed across many compute nodes that can be manipulated in parallel. Spark Core provides several APIs for building and manipulating RDDs.

Spark SQL enables Spark to work on structured data. It allows querying data via SQL and Hive Query Language (HQL), which is an Apache Hive variant of SQL. Different data sources including Hive tables, Parquet, and JSON can be used by Spark SQL. Developers can use SQL queries with RDDs in Python, Java, and Scala, as well as combine SQL with complex analytics all in a single application. However, the abstraction that more naturally fits Spark SQL is offered by DataFrames, which were introduced in version 1.6. At a high level, DataFrames logically represent tables of a relational DB. Previously Spark supported SQL via Shark, a project developed at the University of California, Berkeley, which basically consisted in a port of Apache Hive to run on top of Spark. Shark was later replaced since version 1.0 by Spark SQL, in order to provide better integration with the underlying engine and language APIs.

Spark Streaming enables processing of live data streams such as Internet of Things or message queues containing web services status. Since Spark Streaming's API matches the RDD API in Spark Core, moving between applications that use stored or streaming data is quite easy. Spark Streaming provides the same level of fault tolerance, throughput, and scalability as Spark Core. Furthermore, since Spark 2.0 streaming data, by its nature unbounded and possibly infinite, can be represented and operated upon via the DataFrame abstraction. Users can simply obtain DataFrames from streaming sources, either files or Kafka,[6] and use the same API as with static data.

MLlib is Spark's library for ML. It features different kinds of ML algorithms and tasks, such as classification, regression, and clustering, enabling users to scale out any of these workloads to computational clusters.

GraphX is a library for graph-parallel computation as well as graphs analytics. By extending Spark's RDD API, GraphX allows users to create a directed graph with arbitrary properties in each vertex and edge. Among its capabilities, there are several common algorithms on graphs, like PageRank or triangle counting, alongside basic operators, such as subgraphs or mapping across vertices.

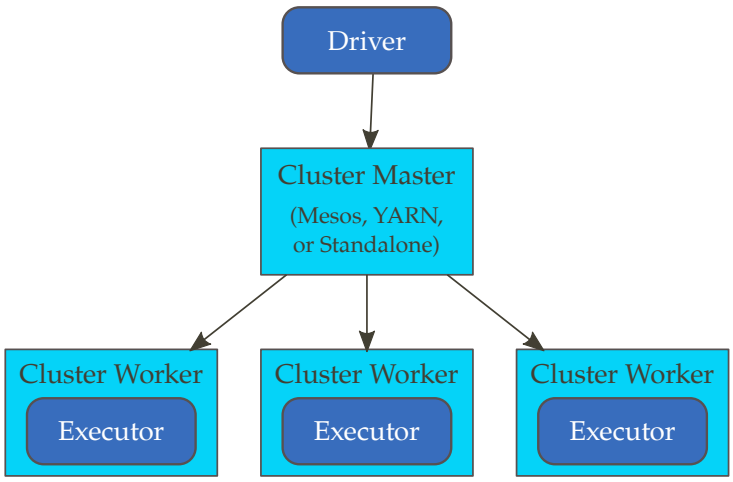


Figure 2.5 – The components of a distributed Spark application

2.1.2.2 Spark Runtime Architecture

When Spark runs in distributed mode, it uses a master/slave architecture with one central coordinator, the *driver*, and many distributed workers, or *executors*. Both the driver and executors have their own separate Java processes. A driver and its executors together form an application.

The Spark driver is responsible for converting user programs into *tasks*, i.e., units of physical execution. We will use RDDs for the following example, in order to keep it simpler. All Spark programs create RDDs from input data, then the API allows either to derive other RDDs via *transformations* or to perform some final operations, called *actions*, which extract some aggregate values or save results. The logical flow is represented as a DAG, where operations are vertices and data dependencies are edges. When the driver runs, it converts this DAG into an execution plan, where all the operations that can be pipelined belong to a single *stage*. Tasks are nothing else than the list of operations that compose a stage applied to a given data chunk. Overall Spark applications can easily require even thousands of tasks.

As soon as the execution plan is ready, the driver schedules ready tasks on executors. Tasks are ready when all their predecessors complete processing and make intermediate data available. In order for the driver to know where to schedule tasks, executors register themselves with it as soon as they are started. Moreover, the driver tries hard to schedule tasks according to data locality, thus placing initial tasks close to a replica of their assigned HDFS block, for instance.

Alongside performing the actual processing, executors also take care of storing into memory RDD blocks, possibly caching them, depending on users' directions and memory pressure. For this reason, executors are started at job submission time and remain active until the application is over. Spark's fault tolerance makes it possible for job to complete even if some executors should

[5] http://mesos.apache.org
[6] https://kafka.apache.org

fail, by rescheduling lost tasks and, if need be, even entire stages to rebuild failed cached blocks.

If executors take care of performing the logical tasks composing an application, instead resources are negotiated by the driver with the cluster manager, which provides a number of *workers* based on job requirements.

2.2 Performance Models for Big Data Applications

Deploying a DIA in a computer cluster requires a non-negligible learning curve of the underlying technology and a good knowledge of the process requirements. It is essential to consider that clusters are shared by multiple users or institutions, are vulnerable to hardware failures and have a monetary cost. The minimization of starvation between user jobs, the impact of execution errors, and the optimization of operational costs are important issues.

Modeling and simulating the configuration of a high performance distributed computer framework allows for predicting DIAs behavior before execution. They enable the detection of potential problems such as bottlenecks, the tolerance to hardware malfunctioning, as well as a finer estimation of the resources usage, the running time, and throughput. Execution time and resources are two of the main parameters for guaranteeing a fair scheduling among user jobs and inferring the billing.

This section is subdivided into three parts, following the three main classes of applications of interest: Apache Hadoop, Spark, and CNNs.

2.2.1 Apache Hadoop

The literature provides a large number of performance studies for Hadoop 1, since the framework has been widely adopted in the ICT industry, often supporting core business activities. Two main approaches have been explored: i) simulation-based models implement the single constituents of Hadoop and of the job, replaying in a simulated environment the steps and delays of the real system; ii) analytical models, on the other hand, define a mathematical representation of those constituents, avoiding the costs of running multiple simulations. Both approaches make use of information such as input dataset size, cluster resources, and Hadoop specific parameters.

Some examples of AMs follow. For instance, Bruneo et al. [26] introduce a stochastic reward Petri net model representing an infrastructure as a service (IaaS) cloud where the load conditions can change dynamically. Performing several analyses on the proposed model, they can assess adaptation strategies with respect to the advantages offered to IaaS providers. Ahmed and Loguinov [2], instead, construct a probabilistic performance model for MapReduce and evaluate it against traces, considering large scale sorting as target workload. On the other hand, Lin et al. [84] define two sets of parameters, one quantifying the capabilities of underlying resources and another collecting jobs characteristics, and propose a system of equations that correlates all those quantities to application performance. Similarly, Yang and Sun [134] formulate a simple model that associates system capabilities and job peculiarities, with a focus on the effect of configuring the number of mappers or reducers of each application on execution times. Another performance model estimating the execution time by considering the single costs of the various phases

of a MapReduce job is described in Lin et al. [85]. In this work, the authors go down to the very low level elements that determine the cost of single job phases, writing a 37-parameter model that provides execution times within 10 % of those measured in a real cluster.

Verma et al. [122] propose the ARIA framework, whose capabilities include estimating the execution time of jobs in MapReduce clusters. Adopting scheduling techniques, the authors prove lower and upper bounds on makespans and derive formulas for performance prediction. Such formulas exploit information contained in the logs produced by similar jobs in previous runs. With this method, they obtain both a conservative estimate, suitable for strict SLAs, and an alternative result that does not guarantee deadlines are met, yet boasts a relative error below 10 % in comparison to measured timings. The same research group later focused on the needs of organizations running data intensive workloads on premises [123] . With a combination of micro-benchmarks, application profiling, and iteratively re-weighted least squares, they obtain a model for the effect of the underlying hardware on jobs performance. The proposed method can help system administrators in choices about hardware upgrades.

Some other approaches, e.g., Liang and Tripathi [81], are based on an approximate mean value analysis technique and use an iterative hierarchical approach. At each iteration, they perform two steps: in the first one, the distribution of task execution times is used to estimate job execution times and the synchronization delay among tasks of the same job. Such information is then used to estimate tasks time overlap to refine the queueing synchronization delay and finally update the mean job execution time estimate. Along the same lines, Vianna et al. [125] combine a precedence graph and a QN to capture the intra-job synchronization constraints, thus being able to estimate the synchronization delays introduced by the communication among mappers and reducers. Unfortunately, even if the approach is rather accurate (around 15 % accuracy on real systems), the authors assume that CPU slots are statically assigned to mappers and reducers, hence the proposed method cannot be adopted to estimate performance under the Hadoop 2 dynamic resource assignment policy.

Several works describe the adoption of Petri nets (PNs) for MapReduce modeling. For instance, Castiglione et al. [28] describe a big data architecture based on Hadoop by means of SPNs and apply mean field analysis to obtain average metrics and estimate its performance. With this approach, the authors obtain a fast approximate method for performance prediction. Another approach, presented by Barbierato et al. [20], exploits generalized stochastic Petri nets alongside other formalisms, such as process algebras or Markov chains (MCs), to develop multi-formalism models and capture HQL queries. Adopting the presented tool, the authors investigate how performance depends on some configuration parameters. In the literature, colored Petri nets (CPNs) have also been adopted to assess the feasibility of a distributed file system project [1]. The authors design a deployment of HDFS exploiting spare resources in a cluster of workstations available for teaching in their university, so as to provide a sufficiently available distributed file system. CPNs are used to assess system availability in a number of configurations of interest. More recently, Ruiz et al. [108] formalized the MapReduce paradigm using prioritized timed colored Petri nets to obtain complete and unambiguous models

of the system behavior. They evaluate the correctness of the system and carry out a trade-off analysis of the number of resources versus processing time and resource cost with CPNTools [66]. Further works with PNs and MapReduce are oriented to measuring performance under failures [68] or studying the fault tolerance mechanism [91].

On the other side, QNs have also been introduced for modeling cloud systems. Bardhan and Menascé [22] apply QN models for predicting the completion time of the map phase of MapReduce jobs within simple configurations of Hadoop. This work is noteworthy as it explicitly considers contention and parallelism on compute nodes to evaluate the execution time of a MapReduce application, yet its weak spot is that it contemplates the map phase alone. Alipour et al. [5] develop a cloud provider independent model with QNs that represents entities involved in the Hadoop MapReduce phases, and customize it for a specific cloud deployment. Finally, in Yu and Li [136] an analytical queueing models has been developed to investigate the utilizations and mean waiting times of mappers and reducers, respectively.

In QN literature, the fork/join paradigm is used to denote the modeling of the concurrent execution of many tasks within higher level jobs [19, 98]. Specifically, this approach operates through two steps: i) jobs are spawned at a fork node in multiple tasks, then ii) they are submitted to queueing stations that, in turn, model the available servers. Once all the tasks have been served, they can synchronize at a join node. It has to be noted that when a fork/join network has more than two queues, a closed form solution is not possible [88]. That said, it is possible to mitigate the issue by using a special kind of structure that considers the MC underlying the QN and representing the possible states of the system [81]. Unfortunately the state space grows exponentially when the tasks number corresponds to realistic MapReduce jobs—in the order of thousands—thus making the above approaches unsuitable [31, 88].

2.2.2 Apache Spark

Apache Spark is a powerful framework oriented to big data processing, which allows users to quickly build applications combining SQL and data analytics. It is gaining wide renown for its improved performance in comparison to Hadoop: Mavridis and Karatza [92] extensively compare the two frameworks when applied to web log files analytics, thus proving a large speedup when similar queries are executed on Spark. However, Gu and Li [57] study the impact of dataset size on performance and argue that, despite the large speedup when datasets fit nicely into cluster memory, Spark suffers a stronger degradation when the dataset grows and in memory caching loses effectiveness. These considerations motivate the adoption of performance modeling techniques, which could offer valuable insight into such issues, as well as provide accurate predictions in configurations where the use of Spark is advantageous.

The picture is further complicated by the effects of the full software stack and underlying hardware on observed performance. Chiba and Onodera [30] study in depth some queries from the TPC-H benchmark, considering logs, performance counters, Java virtual machine and OS level profiling, in order to identify bottlenecks and non-optimal behavior. Exploiting this information, they optimize the configuration across the full stack and achieve a 30–40 % speed increase on average, yielding up to 5x faster execution over default.

Wang and Khan [127] model Spark jobs performance as a system of algebraic equations and propose to profile them via small scale experiments. The evaluation is conducted with several well known applications, such as PageRank and k-means, and suggests that scaling down both the dataset and cluster sizes can lead to inaccuracies for the prediction of I/O. The same research group later extended this model by taking into account the effects of interference and contention in virtualized cloud environments [128]. As different applications show different behavior patterns, alongside an improved profiling framework they also propose an algorithm to assess interference even when it is due to a cascading effect.

Islam et al. [64] propose to model Spark application performance via a set of power laws. These are obtained by profiling the job on a portion of the input dataset, and then used as underlying performance model for a heuristic algorithm that searches the minimum cost deployment while satisfying a deadline on execution time. The validation shows that the technique allows for saving in comparison to an over-provisioned cluster.

Recently, given the framework complexity, black box approaches based on ML have been proposed to predict Spark applications performance as a function of dataset size, executors memory, and number of cores [54].

2.2.3 Deep Learning and GPUs

Deep learning's popularity is steadily increasing thanks to its impact on many application domains, ranging from image and voice recognition to text processing, and has received a lot of interest from many academic and industry groups. Advances are boosted by enhancements of the deep networks structure and learning process (e.g., dropout [114], network in network [82], scale jittering [126]) and by the availability of GPUs, which allows to gain up to 40x improvement over traditional CPU systems.[7]

Over the last few years, several frameworks have been developed and are constantly extended to ease the development of DL models and to optimize different aspects of training and deployment of DL applications. Bahrampour et al. [18] provide a comparative study of Caffe, Neon, Theano, and Torch, by analyzing their extensibility and performance and considering both CPUs, when possible in a multi-threaded setting, and GPUs. The paper provides insights on how performance varies across input batch size and different convolution algorithm implementations, but it does not provide any means to generalize performance estimates to different settings.

Due to the complexity of big data and DL frameworks, where multiple software stacks are involved and/or the level of parallelism is very large, black box performance modeling approaches based on ML are recently favored against more traditional AMs, like, e.g., QNs or PNs. For instance, Hadjis et al. [58] developed solutions to minimize the total training time of CNNs, given the network architecture of the DL model, the dataset target of the training, and the set of available computational resources, characterized by the number of CPUs/GPUs available on machines, their throughput, and the network speed.

[7] http://www.nvidia.com/object/why-choose-tesla.html

Since commonly CNNs are deployed on GPUs, it is relevant to understand how such hardware can influence performance. Jia et al. [67] propose the Stargazer framework to build application-specific performance models that correlate execution time and several GPU parameters. Given the daunting size of the design space, they exploit sparse random sampling to obtain a small, yet representative, dataset and then apply an iterative model selection procedure to determine the most important predictors, thus creating step by step an accurate linear regression model.

Another approach to the issue is proposed by Liu et al. [86], who elaborate a detailed representation of general purpose applications on GPUs. After categorizing the subtasks possibly run on the GPGPU into data constant, data linear, and computation dependent, they provide three general expressions and the relevant parameters for all the main phases of GPU computation. With these, they achieve an accurate performance model, validated against a bio-sequencing application.

Kerr et al. [72] profile and build models for a range of applications, run either on CPUs or GPUs. Relying on 37 performance metrics, they exploit principal component analysis and regression in order to highlight those features that are more likely to affect performance on heterogeneous processors. Along the same lines, Luk et al. [87] develop Qilin, a technique for adaptively mapping computation onto CPUs or GPUs, depending on application as well as system characteristics. With this approach, they show an improved speedup with respect to manually associating jobs and resources.

2.3 Hybrid Machine Learning Approaches

In recent years, machine learning became popular to predict the performance of complex computer systems. Ipek et al. [62] adopted artificial NNs to predict the impact of architectural changes on performance metrics while studying memory systems and multi-threaded CPUs. Yigitbasi et al. [135] compared several ML methods to predict Hadoop clusters performance, ranging from ordinary linear regression to advanced techniques like artificial NNs, regression trees, and support vector regression (SVR) with diverse MapReduce applications and cluster configurations. Lama and Zhou [76] proposed AROMA, a system based on SVR for automatic resource allocation and configuration in cloud-based MapReduce clusters. AROMA mines historical execution data in order to profile past submissions and to match incoming jobs to the available performance signatures for prediction. In this way, the proposed system can avoid violations of the deadlines stated in SLAs, while incurring minimum cost, with an average percentage error on running time predictions around 20 %.

Black box modeling can derive performance models from data to make predictions without a priori knowledge about the internals of the target system. In this research area, Venkataraman et al. [121] built Ernest, a black box performance prediction framework for large scale analytics based on experiment design to collect the minimum number of training points. The approach was evaluated by predicting the performance of different business analytics workloads using Spark MLlib on Amazon EC2, achieving average prediction errors under 20 %. Recently, Ernest has been integrated within the Heming-

way framework [100] with the aim to identify the optimal cluster configuration for ML training algorithms, where the convergence rate may be affected by the cluster size. Ernest's computational model was used to estimate the time taken per iteration as a function of the cluster size. Hemingway uses linear regression to estimate how the convergence rate changes with the number of involved machines.

Along the same lines, Alipourfard et al. [6] proposed CherryPick, a black box system leveraging Bayesian optimization to unearth the optimal or near-optimal cloud configurations that minimize cloud usage cost while guaranteeing application performance. The authors' approach limits the search overhead for recurring big data analytics jobs, focusing the search to improve prediction accuracy of those configurations close to the best for a specific deadline. This can be a limit in more general settings, for instance under QoS constraints or loads that vary over time. Delimitrou and Kozyrakis [39] proposed Quasar, a black box cluster management system that maximizes resource utilization while meeting performance and QoS constraints. The authors exploit classification techniques to determine the impact of the type and amount of resources, as well as of workload interference, on system performance.

Some more recent works exploited the possibility to use AMs and ML in synergy to get the best of both worlds. Tesauro et al. [116] proposed autonomic resource allocation in a multi-application prototype data center with the goal of maximizing the total expected business value. They show how to combine the strengths of both reinforcement learning (RL) and QNs in a hybrid approach, in which RL trains offline on collected data while a queuing model policy controls the system. Thereska and Ganger [117] presented a hybrid performance modeling framework, which uses the redundancy of high level system specifications described through models and low level system implementation to localize system-model inconsistencies and give hints to the system and model designer regarding the root cause of the problem. In this work, mathematical models based on queueing theory are coupled with regression trees. Herodotou et al. [59] proposed Elastisizer, a system to which users can express cluster sizing problems as queries in a declarative fashion. In this system, the overall process of estimating execution time and cost of MapReduce jobs is broken down into four smaller steps and, for each step, a suitable white box or black box modeling approach is chosen.

There are also some works that target in memory transactional data stores: Rughetti et al. [107] used a mixed AM/ML approach to dynamically tune the level of concurrency of applications based on software transactional memory to optimize system throughput. The AM and ML techniques used in this research are parametric analytical modeling and NNs, respectively. Didona et al. [47] introduce Transactional Auto Scaler (TAS), a system for automating the scaling of fully replicated in memory transactional data grids in cloud platforms. In TAS, analytical and ML models were incorporated to predict throughput, commit probability, and average response time. White box models, based on queueing theory, are used to capture the dynamics of concurrency control or replication algorithms, so as to forecast the effects of data contention, as well as the effects of contention due to CPU utilization. Didona et al. [43] consider the issue of automatically identifying the optimal degree of parallelism of an application using distributed software transactional memory by introducing a hybrid approach. They exploit TAS [47] as the analytical

performance model, while decision tree (DT) regression is utilized as the ML technique.

Didona and Romano [46] investigate a technique whose main idea consists in relying on an AM to generate a knowledge base (KB) of synthetic data over which a complementary ML is initially trained. The initial KB is then updated over time to incorporate real samples from the operational system. For updating the KB, the authors propose different algorithms based on merge and replacement. As case studies, the authors consider Infinispan and total order broadcast relying on DT regression and queueing models. The effect of the proposed parametrized algorithms on the mean average percentage error (MAPE) of the gray box model is evaluated by means of ten-fold cross validation. However, the authors do not investigate when the algorithm should stop updating the ML model.

2.4 Capacity Planning and Management of Big Data Frameworks

Capacity planning and architecture design space exploration are important problems analyzed in the literature [4, 25]. High level models and tools to support software architects (see, e.g., Palladio Component Model and its Palladio Bench and PerOpteryx design environment [23, 73], or stochastic process algebra and the PEPA Eclipse plugin [97, 119]) have been proposed for identifying the best configuration given a set of QoS requirements for enterprise web-based systems, but unfortunately they do not support cloud-specific abstractions or (see, e.g., Kross and Krcmar [75]) directly address the problem of deriving an optimized cloud and big data cluster configuration.

On the other side, capacity management, cluster sizing, and tuning of big data applications have received also a widespread interest by both academia and industry. Provisioning and scheduling resources for big data applications in cloud infrastructures face several challenges such as query dynamicity, load fluctuations, performance unpredictability, and resource heterogeneity. One of the main challenges for big data cluster frameworks is how to partition and dynamically allocate resources to reach high efficiency and scalability. Resource partitioning and dynamic allocation mechanisms, indeed, are enablers for providing efficient resource provisioning and improving system utilization. Recently, significant work was performed to address these issues, also decoupling resource management from the programming model: a number of technologies have been proposed, such as YARN [120], Mesos [61], Omega [111], and Borg [124].

Big data frameworks often require an intense tuning phase in order to exhibit their full potential. For this reason, Herodotou et al. [60] propose Starfish, a self-tuning system for analytics on Hadoop. In particular, Starfish collects some key run time information about applications execution with the aim of generating meaningful application profiles; such profiles are in turn the basic elements to be exploited for Hadoop automatic configuration processes. Furthermore, also the cluster sizing problem has been tackled and successfully solved exploiting the same tool [59]. More recently, Dalibard et al. [35] have presented BOAT, a gray box framework, which supports developers to build efficient auto-tuners for their complex computer systems, in

situations where general purpose auto-tuners fail. BOAT is based on structured Bayesian optimization and has been used to support the performance tuning of Cassandra clusters and of GPU-based servers for NN computation, even with heterogeneous resources.

The problem of progress estimation of multiple parallel queries is addressed in Morton et al. [95]. To this aim, the authors present Parallax, a tool able to predict the completion time of MapReduce jobs. The tool has been implemented over Pig, while the PigMix benchmark has been used for the evaluation. ParaTimer [94], an extension of Parallax, features support to multiple parallel queries expressed as DAGs.

The capacity management and cluster sizing problems, instead, have been faced by Tian and Chen [118]. The goal is the minimization of the execution cost for a single MapReduce application. The authors present a cost model that depends on the dataset size and on some characteristics of the considered application. A regression-based analysis technique has been used to profile the application and to estimate model parameters.

MapReduce cluster sizing and scheduling is considered in Lin et al. [83]. The authors propose a tandem queue with overlapping phases to model the execution of the application and an efficient run time scheduling algorithm for the joint optimization of the map and copy/shuffle phases. The authors demonstrated the effectiveness of their approach comparing it with the offline generated optimal schedule.

Another work by Curino et al. [34] proposes an ad hoc language to reserve resources on a shared YARN cluster. Knowing current and future resource requirements, the authors develop a mixed integer linear programming formulation for the resource allocation and scheduling problem, which then they solve exploiting two novel heuristics. The experimental validation shows that the proposed approach enables full cluster utilization, with SLAs that are met for all the accepted jobs and a throughput improvement of 15 %. In a similar direction, Jyothi et al. [69] propose to automatically extract service level objectives from historical data, in order to schedule in advance periodic jobs. This goal is achieved by placing recurring reservations and using linear programming to efficiently pack jobs, thus retaining high cluster utilization while also obtaining fewer violations. The proposed Morpheus framework has also a dynamic re-provisioning component to mitigate the sources of performance unpredictability.

Cluster sizing based on deadlines for MapReduce applications is considered in Phan et al. [101]. The authors recognize the inadequacy of Hadoop schedulers released at the date to properly handle completion time requirements. The work proposes to adapt to the problem some classical multiprocessor scheduling policies; in particular, two versions of the earliest deadline first heuristic are presented and proven to outperform off-the-shelf schedulers. A similar approach is proposed in Zhang et al. [140], where the authors present a solution to manage clusters shared among Hadoop application and more traditional web systems. Zhang et al. [140] investigate the performance of MapReduce applications on homogeneous and heterogeneous Hadoop clusters in the cloud. They provide a simulation-based framework for minimizing cluster infrastructural costs, yet a single class of workload is optimized.

The ARIA framework [122] addresses the problem of calculating the most suitable amount of resources to allocate to map and reduce tasks in order to

meet a user-defined due date for a certain application: the aim is to avoid as much as possible costs related to resource over-provisioning. The same authors, in a more recent work [142], provided a solution for optimizing the execution of a workload specified as a set of DAGs under the constraints of a global deadline or budget. This work considers heterogeneous clusters with possible faulty nodes, too.

Mian et al. [93] provide a framework facing the problem of minimum cost provisioning of MySQL clusters in cloud environments. The cost model includes resource costs and SLA penalties, which are proportional to execution time violations of a given deadline. Queries execution times are predicted through QN models, which, however, introduce up to 70 % relative errors. The minimum cost configuration is identified via two greedy hill climbing heuristics, which can identify heterogeneous clusters, but no guarantees on the quality of the final solution can be provided. Delimitrou and Kozyrakis [38] provide a run time framework for the management of large cloud infrastructures based on collaborative filtering and classification, which supports run time decision of a greedy scheduler. The overall goal is to maximize infrastructure utilization while minimizing resource contention, taking into account also resource heterogeneity: this goal is reached via exploiting information from previous runs and offline training, so as to predict performance and interference effects. The same authors extended their work in Delimitrou and Kozyrakis [40], supporting resource scale-out decisions (i.e., determining if more servers can be beneficial for an application) and server scale-up (i.e., predicting if more resources per server are beneficial) for Spark and Hadoop applications. If performance deviates from SLAs, Quasar reclassifies the workload and readjusts resource allocation and/or assignment, so as to meet the deadlines with minimum costs. The authors demonstrated that their framework can manage effectively large systems, significantly improving infrastructure utilization and application performance. If their collaborative filtering approach requires to gather little data from the running applications, in turn it requires a significant effort to initially profile the baseline benchmarking applications used to predict the effects of, e.g., resource contention and scale-up/out decisions at run time: the exhaustive profiling of 30 workload types running from 1 to 100 nodes.

2.5 Open Issues

After exploring the state of the art in the main directions of performance modeling and resource allocation, this final section wraps up the exposed content and draws some observations on the relevant aspects that still need to be addressed.

When dealing with DIAs performance, the literature studies feature three main approaches. Some proposals build formulas that take into account the fundamental constituents of jobs, parametrize them via benchmarks or profiling, then predict performance metrics. More convenient alternatives are offered either by other works that describe AMs based on well known formalisms such as PNs, QNs, and MCs, or by proposals adopting ML techniques to derive models. In this comparison, AMs are quite insensitive to the configuration range where to predict, but suffer from possibly long simulation

times. Depending on the constraints on prediction times, this might call for simplifying assumptions, at the expense of accuracy, or even rule out some formalisms outright. On the other hand, ML enables fast prediction, but requires costly experiments to collect training data and can lose accuracy, even dramatically, outside of the training domain.

With the aim of addressing the mentioned drawbacks, two principal classes of approaches are currently explored: one is the use of optimal experiment design techniques to minimize data collection and, consequently, experimental costs, without negatively affecting accuracy; another possibility is the adoption of hybrid methods, in order to exploit the advantages of both basic techniques and compensate the shortcomings. Along the lines of research question 1, this dissertation considers mainly AMs and hybrid approaches, trying to identify the most fitting methods based on the requirements to satisfy, in terms of accuracy or speed.

Performance modeling for DL frameworks offers a less varied scene: these applications boast an extremely high level of parallelism, so generally the literature in the field does not contemplate AMs, given the impractical simulation times. Several proposals use benchmarks or profiling, while others adopt ML to associate system capabilities and application characteristics with observed performance. We tread the latter path when proposing models for Caffe, the DL framework.

The other relevant part of this dissertation is cluster sizing and run time resource management. These problems are frequently solved, in the literature, considering simplified scenarios. For instance, some works focus on specific workload classes and lose generality, while other proposals cannot handle concurrent users, but only single user scenarios. Other drawbacks that can be noticed in available solutions are the need for huge benchmarking/profiling campaigns, or the applicability only to recurrent jobs, with the same application that runs repeatedly with the same deadline. In answering research question 2, our goal is to overcome most of these shortcomings, obtaining general techniques able to predict performance under varying concurrency levels or SLAs. Particularly in the design time setting, we aim at containing the required profiling campaign: running many experiments at the real scale would defeat the purpose of design tools.

CHAPTER 3

Performance Models

This chapter provides an overview of several performance models developed for a gamut of big data applications: we propose models for Hadoop/MapReduce and Apache Spark, on public and private cloud clusters, as well as CNNs. Initially the focus was on Apache Hadoop for its role as enabling technology for big data, but as soon as Spark took over due to its advantages in terms of performance and flexibility we also moved our attention to the more recent framework. Further, a collaboration with IBM Research brought about novel modeling approaches and some interesting findings on CNNs' performance.

Performance modeling covers a major role both for design considerations and for resource management at run time, yet it is usually a hard task to accomplish. To date, big data systems are massively parallel and show complex behaviors due to precedence constraints, data dependencies, and strict synchronization points, with further complexity added by the effects of the underlying hardware. Notwithstanding the difficulty of devising appropriate performance models, alternative approaches based on extensive experimentation are impractical and anti-economical. These considerations motivate the investigation of various formalisms with different levels of expressiveness and efficiency, so as to explore several trade-offs between accurate predictions and fast solutions. In particular, formalisms like QNs and SWNs are natural candidates for modeling the kind of distributed frameworks involved in big data applications, since they provide a fork/join mechanism that closely matches the division of work into a number of data-parallel tasks widely used in Hadoop and Spark. These representations allow for a great deal of accuracy, but require time consuming simulations, too. Moreover, such expressive models can easily be enriched so as to investigate some specific scenarios, e.g., reliability or failure recovery. On the other hand, models obtained via the application of ML techniques offer significantly faster predictions at the expense of accuracy and extensibility. These specificities led us to use ML-derived models in finding initial guesses for the optimization procedure, whilst simulation-based approaches make for accurate prediction devices in the final, heuristic phase.

In the following Hadoop 2 is adopted as reference example to discuss per-

formance models. This choice is motivated by the many similarities between Apache Hadoop and Spark: qualitatively the two frameworks largely share a common basic structure, whilst the main difference lies in the more general program flow allowed by Spark's in memory abstractions. However, adapting the proposed models to DAGs requires only the repetition of some already present blocks. Practically, both the map and reduce phases in Hadoop models are represented with the same sub-network, then stages in Spark DAGs can easily be modeled in the same way.

In the following, Section 3.1 lists some basic assumptions used in deriving performance models for Apache Hadoop and Spark. Sections 3.2 and 3.3 detail models for Hadoop 2 with two possible formalisms, queueing networks and stochastic well formed nets. Later on, Section 3.4 describes the use of fluid techniques and how to generalize any of the preceding models to Apache Spark. Section 3.5 focuses on performance modeling for CNNs via the adoption of linear regression, while Section 3.6 presents an algorithm for hybrid ML. At last, Section 3.7 wraps up this chapter by discussing pros and cons of all the described techniques, with a particular focus on optimization, which is the topic of the next chapter. On top of this, the final section also shows how this chapter contributes to answering research question 1.

3.1 Modeling Assumptions

Modeling the performance of Hadoop 2 clusters is challenging since, differently from the previous release where resources, i.e., CPU slots, were statically split for mappers and reducers, in the latest Hadoop containers are assigned dynamically among ready tasks, leading to a better cluster utilization. In particular, we focus on clusters governed by the Capacity Scheduler, which allows for partitioning the cluster among multiple customers through queues, each queue being regulated by a FIFO policy. In the following we assume that queues are partitioned and hence we can focus on single class systems.

The parallel execution of multiple tasks within higher level jobs is usually modeled in the QN literature with the concept of fork/join: jobs are spawned at a fork node in multiple tasks, which are then submitted to queueing stations modeling the available servers. After all the tasks have been served, they synchronize at a join node. Unfortunately, there is no known closed form solution for fork/join networks with more than two queues, unless a special structure exists [81]. Hence, the performance metrics of such networks must be computed by considering the MC underlying the QN, which represents the possible states of the system [81]. However, such approaches are not fit for Hadoop systems, since the state space grows exponentially with the number of tasks [31, 88], in the order of thousands in realistic MapReduce jobs and tens of thousands in Spark jobs. For this reason, a number of approximation methods have been proposed. In particular, Nelson and Tantawi [96] proposed a good approximation technique that, however, is based on service times with exponential distribution, which is not the case for Hadoop deployments. Our initial experiments showed that mapper and reducer times follow general distributions, which can be approximated by phase type or in some cases Erlang. Under the exponential time hypothesis, the relative error observed in our simulations was around 50–60 %.

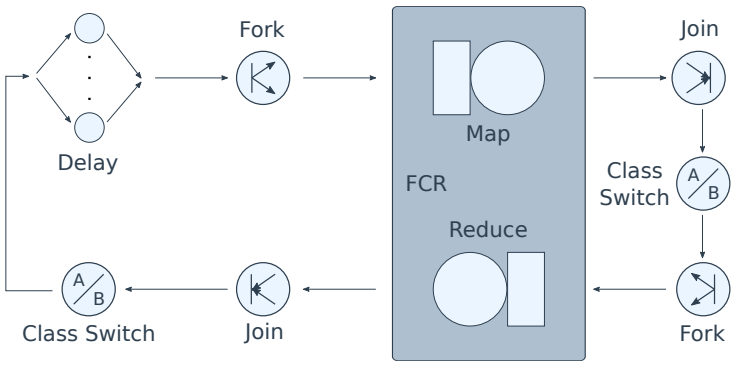


Figure 3.1 – Queueing network model

For this reason, we developed simulation models based on the concept of finite capacity region (FCR) available in modern QN simulators [24]. Unfortunately, QN models capture the behavior of Hadoop 2 with some approximations. In Section 3.3 we rely on SWNs and provide a model that better captures the behavior of real Hadoop 2 systems. Moreover, we investigate an advanced cloud-based scenario where some resources are provided by unreliable spot instances and we evaluate the performance of jobs in case of failure.

3.2 Queueing Network Model

This section discusses a proposal of QN model for MapReduce applications running upon YARN Capacity Scheduler. The performance model is depicted in Figure 3.1. It is a closed QN model where the number of concurrent users is given by H and they start off in the delay center, characterized by the average think time Z. When a user submits her job, this is forked into as many map task requests as stated in the job profile, which then enter the FCR. FCRs model situations where several service centers access resources belonging to a single limited pool, competing to use them. Hence, the FCR enforces an upper bound on the total number of requests served at the same time within itself, allowing tasks to enter according to a FIFO policy, but also supporting prioritization of different classes. The FCR includes two multi-service queues that model the map and reduce execution stages. The FCR and multi-service queues capacities are equal to the total number of cores available in the cluster. In this way, we can model the dynamic assignment of YARN containers to map and reduce tasks whenever they are ready. Map tasks are executed by the first multi-service queue and synchronize after completion by joining back to a single job request; the reduce phase is modeled analogously. Note that the map join is external to the FCR in order to model that when map tasks complete they release container cores, which can be assigned to tasks ready in the FCR FIFO queue. Moreover, the reduce fork is also external to the FCR to model correctly applications characterized by a number of reducers larger than the total cluster capacity.

YARN Capacity Scheduler implements a FIFO scheduling policy within the same queue and containers are allocated to the next job only after all reduce tasks have obtained their resources. The class switches present in the

QN are meant to enforce that reduce tasks waiting for resources obtain them with priority. Despite this, the model in Figure 3.1 is still an approximation: notwithstanding the higher priority associated to reducers, subsequent users' mappers can still occupy part of the servers available in the FCR when the preceding job has an overall number of map tasks that is not multiple of the cluster capacity. In such a case, the last map wave leaves room for serving further requests, hence the following user can overtake part of the capacity and the reduce stage of the first user will not start processing at full capacity until those mappers complete.

Note that the discussed model has a single application class, like the subsequent SWNs. Even if both formalisms can express multiple application classes and possibly different schduling policies (e.g., fair scheduling or work-conserving schemes), we do not present any such model due to the impractical simulation times. Indeed, preliminary analyses demonstrated that only two classes are enough to cause state space explosion, leading to some days (sometimes weeks), per simulation. In the perspective of exploiting them for the optimization procedures in Chapter 4, we prefer to stick to single-class models and heuristically separate the multi-class mathematical programming formulations.

As a final consideration, note that the discussed model is rather general and can be easily extended to consider also Tez or Spark applications, where a Tez DAG node or Spark stage is associated to a corresponding multi-server queue and fork/join pair.

3.3 Stochastic Well Formed Net Models

In this section we present two models based on the SWN formalism. The first one, in Section 3.3.1, is quite similar to what discussed in Section 3.2: a basic performance model for DIAs on the Capacity Scheduler, but the greater expressiveness of SWNs allows for matching more closely the real scheduling behavior. Specifically, if the model in Figure 3.1 cannot enforce the FIFO queue of jobs, SWNs are powerful enough to model this mechanism. Later on, Section 3.3.2 describes a more detailed model that can be used to assess the detrimental effects of spot VMs failures, both in terms of delays in the execution time and costs.

3.3.1 Basic Performance Model

The SWN in Figure 3.2 is able to capture completely the behavior of the Capacity Scheduler policy. Jobs execution is modeled by a closed workload, where the $nU1$ users compete to access the cluster and cycle between demanding to execute the MapReduce scenario (subnet in the dotted rectangle), and spending an external delay period between the end of one response and the next request (mean firing time of the *think* transition). The basic color class *User* consists of a single subclass *User1* and the job identities are captured by assigning a token of different color to each job $u_1, \ldots, u_{nU1}$.

To enforce the FIFO scheduling each job is assigned an ID i. The initial marking *M3* of the place *IDs1* is set to the first index of the color class *ID*, and once transition *think* sends a job to the ready state it increases this index by

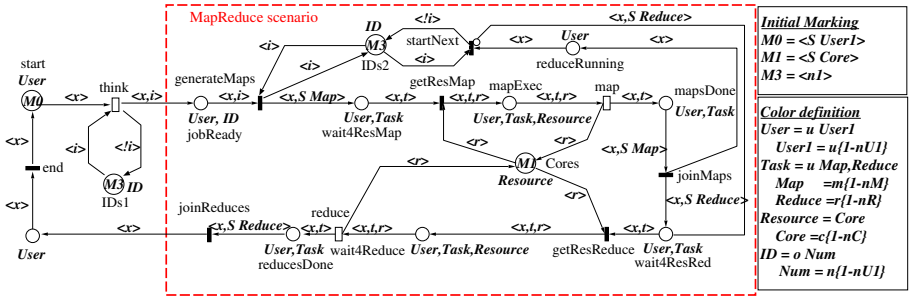


Figure 3.2 – Basic SWN model

one. The transition *generateMaps* will start the job that has the index equal to the one it is getting from place *IDs2*. In other words, the job that has its turn will start the map phase. Whenever a job gets resources for all of its reduce tasks (place *wait4ResRed* drains), the job with the next index will be started thanks to the transition *startNext*, which updates the *IDs2* place with the next index.

When a job x is ready to be processed and it has its turn—i.e., the place *jobReady* is marked with a token $\langle x, i \rangle$ and the *IDs2* place is marked with the same index i—nM map tasks are generated (firing of *generateMaps* transition). Such tasks, associated to job x, are represented by nM pairs $\langle x, t \rangle$, where the color t belongs to the subclass *Map* of the basic color class *Task*. Each task $\langle x, t \rangle$ needs to acquire a resource r to be executed (firing of *getResMap* transition) and map tasks can be concurrently executed according to resource availability. The set of resources is defined by the basic color class *Resource*, which consists of a unique partition *Core* including nC resources. The timed transition *map* models the duration of the map task execution and its firing time is an Erlang-distributed random variable. The map stage is finished when all the map tasks $\langle x, t \rangle$ associated to job x have been executed: the firing of the *joinMaps* transition models the beginning of the next processing step, where nR reduce tasks are generated. The reducing step is similar to the mapping step, the only difference is that the reduce tasks, associated to job x, are represented by nR pairs $\langle x, t \rangle$ where the color t belongs to the subclass *Reduce* of the basic color class *Task*. Finally, observe that the map tasks $\langle y, t \rangle$, associated to a job y, are generated when all the reduce tasks $\langle x, t \rangle$, associated to the previous job x, are not waiting for resource availability. This condition is modeled by the inhibitor arc inscription from place *wait4ResRed* to transition *startNext*.

3.3.2 Performance Degradation with Unreliable Resources

The models considered so far can capture the behavior of MapReduce systems running on an enterprise infrastructure or public clouds based on standard resources. Cloud providers (see, e.g., Amazon EC2[1]) offer another type of resource loaning, *spot*, in which customers bid for the instance price. The VM instances are billed the spot price, which is set by the infrastructure provider and fluctuates periodically depending on current energy costs, which vary

[1] https://aws.amazon.com/ec2/spot/

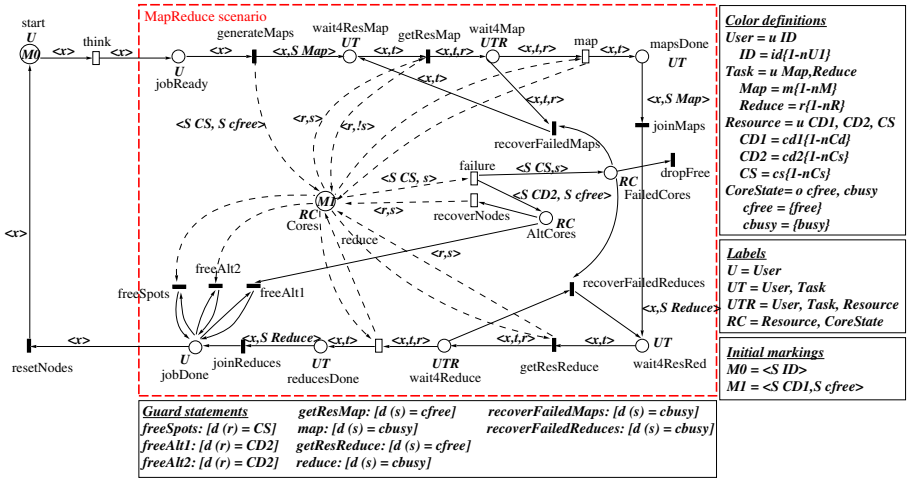


Figure 3.3 – SWN model with spot resources

throughout the day, and also on the overall supply and demand of virtual resources. Spot instances are usually available at a competitive price. However, if the provider raises the spot price above user's bid while she is using spot instances, these can be arbitrarily terminated without notice. Hence, on one side spot resources are an opportunity for lowering the execution cost of MapReduce applications, but on the other side they introduce availability threats.[2]

The model introduced in Figure 3.3 allows for exploring different configurations and to evaluate MapReduce jobs performance degradation in case of spot instances termination. In particular, we assume that once spot VMs fail, a monitoring component replaces them with the same number of on demand resources. Since we are interested in the performance degradation of the job currently in execution, we can drop the FIFO mechanism from the model of Figure 3.2, i.e., the places *IDs1*, *IDs2*, and *reduceRunning*; the transition *startNext*; and the color class *ID*. The model in Figure 3.3 introduces also a new color class *CoreState*, which consists of two singleton subclasses *cfree* and *cbusy*, to record the state of a node. To enable this, the color domain of the place *Cores* is set to the Cartesian product $Resource \times CoreState$. This change enables the place *Cores* to track the occupied cores as well as the free cores. Transitions *getResMap* and *getResReduce* are modified to change the core status from "free" to "busy" whenever they own a core, and vice versa for transitions *map* and *reduce*.

The color class *Resource* is enriched with two new subclasses *CoreSpot (CS)* and *CoreDemand2 (CD2)*, which model spot instances and the on demand nodes triggered to replace spot resources. The color definition *Core* is renamed to *CoreDemand1 (CD1)* to model the on demand nodes that are initially

[2]Spot instances used to behave as explained here at the time of writing the relevant publication, Ardagna et al. [9]. Currently the price is fixed and Amazon can terminate these kind of instances on a short notice. The availability assessment remains valid despite this change, as the discussed model focuses on unpredictable failures, rather than on the details of the pricing mechanism.

started together with spot resources. The places *FailedCores* and *AltCores* are added to identify the failed nodes and the alternative nodes waiting for recovery. The timed transition *failure* removes all the spot instances from the available nodes with a rate proportional to the failure probability. When spot instances fail due to a low bid, it takes at most a YARN heartbeat for the monitoring component to figure out the loss. After this short delay, the replacing process starts acquiring on demand nodes. As soon as the new on demand VMs are ready with running NodeManagers, they have to be registered with the ResourceManager in order to be used by the running job. We summed up all these delays and introduced the timed transition *recoverNodes*, characterized by an appropriate rate. Moreover, we included three instantaneous transitions (*recoverFailedMaps*, *recoverFailedReduces*, and *dropFree*) to move a failed map or reduce task to the waiting list and to drop the failed spot nodes that were not occupied by any task.

Since our goal is to evaluate the average performance of the job when a failure happens (note that a failure can occur anytime between the start and end of the job execution and in the latter case the job might complete before new on demand VMs become available), we have to create the same environment for every successive run of the job. Then, three transitions *freeAlt1*, *freeAlt2*, and *freeSpots* are added to free the places *AltCores* and *Cores* from any residual *CD2* and *CS* at the end of the job execution and one outgoing arc is connected to the transition *generateMaps* to put back the spot cores in the place storing available nodes. As a result, spot failures can happen again while the simulator is running the next iteration. In this way we ensure that every job submission is subject to failure and obtain relevant statistical results.

3.4 Fluid Models

Very often modelers are bound to face several problems when trying to provide a description of a physical system by using a formalism. Typically, these problems are more evident in techniques using discrete states, since the analysis produces a state space explosion, with an exponential growth in the number of states following the complexity of the model. Different alternatives to mitigate this problem have been proposed: in particular, fluid techniques involving a continuous and discrete part have proved to reduce significantly the severity of the issue. Continuous variables can be exploited in different scenarios: for example, they can be used to represent the rising temperature in a closed room or the water leaking out of a full bucket.

In literature, fluid models have been presented in different ways for what concerns the realization of the continuous aspect. Among the different flavors, Gribaudo and Telek [56] refer to i) fluid stochastic Petri nets (FSPNs), ii) reward, and iii) fluid models. FSPNs add to SPNs introducing the ability to handle continuous parts. In reward models, the idea consists in using a MC and associating a reward rate, a positive weight whose value depends on the time spent in a particular state. Considering a specific time interval, it is possible to aggregate all the gathered rewards into a continuous variable. Fluid models proper are a generalization of the reward case, with fluid rates that can be negative and additional constraints on the overall fluid level, such as keeping it nonnegative or imposing an upper bound on it. Fluid models

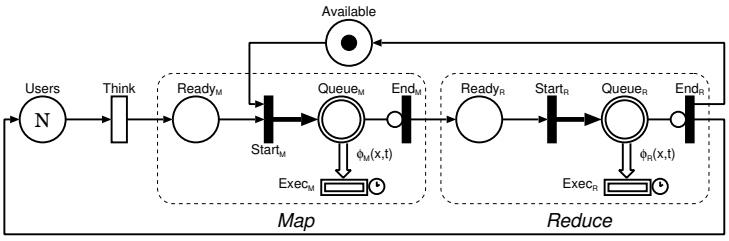


Figure 3.4 – FSPN model of a MapReduce job

can be used to successfully study the MapReduce framework and even more sophisticate ones, such as Spark [138], including paradigms like concurrent programming.

In the following, Section 3.4.1 describes the FSPN we proposed, while Section 3.4.2 describes how to generate an approximation of the average execution time of jobs in the deterministic limiting case, while Section 3.4.3 considers the exponential limiting case. Section 3.4.4 considers a convex combination of the previously presented limiting cases; finally, Section 3.4.5 adds Spark generalization.

3.4.1 FSPN Model for a MapReduce Job

In order to formalize the proposed fluid model, let us focus on the MapReduce paradigm, and describe it using the FSPN shown in Figure 3.4. In Section 3.4.5 the model will be generalized to Spark jobs. Note that the purpose of using a FSPN model is just to specify the underlying fluid model without enumerating its states: for this reason we will just give a brief description of the conventions adopted in the formalism.

In this formalization, single circles represent *discrete places*, which can hold an integer number of tokens; single boxes represent *timed transitions* that can fire after an exponential amount of time; bars represent *immediate transitions*, which fire as soon as they are enabled; thin lines define *standard arcs* that move tokens among places and enable the connected transitions; and thin lines with circular ends define *inhibitor arcs*, which prevent the corresponding transitions from firing whenever the input place is marked. Double circles represent *fluid places*, which can hold a continuous amount of fluid; double boxes identify *fluid transitions*, which continuously pump fluid in and out of continuous places; double arrows represents *fluid arcs* that can remove fluid from their input places; and thick arrows represents *set arcs*, that immediately insert a given amount of fluid in their destination place when their input transition fires.

The discrete amount of tokens is moved across the discrete arcs as usual. With regard to the fluid places, they include a level denoted by a continuous variable, which flows according to an instantaneous rate. The discrete FSPN component regulates the fluid flow through the continuous part, while the conditions enabling a transition depend on the discrete component only.

The model of the MapReduce paradigm considers N users (corresponding to the marking of place `Users`) that can submit jobs to the system after a think time Z. The submission of jobs from a user is modeled by the firing

of transition *Think*, characterized by the infinite server semantic. According to the YARN Capacity Scheduler, we consider that only one job at a time can be executed by the system: this is obtained via place *Available*, which holds a token whenever the infrastructure is ready to run a new job, thus enabling the corresponding *Start* transition. Both the map and reduce phases (as well as generic Spark stages) are modeled by a similar sub-network. Place *Ready* holds a token whenever the phase is ready to start, and its beginning is modeled by the firing of the immediate transition *Start*. This transition inserts, via its output set arc, the number of tasks (either map or reduce) that need to be executed in the corresponding phase in fluid place *Queue*. The execution of the jobs is modeled by the time-dependent fluid transition *Exec*, which is connected to place *Queue* with an output arc. Transitions *Exec* have a special time-dependent semantic that will be described in the following sections: for this reason it is represented with a small clock drawn at its side. Whenever the queue is empty, either the next phase can start or the job can end thanks to the firing of transition *End*.

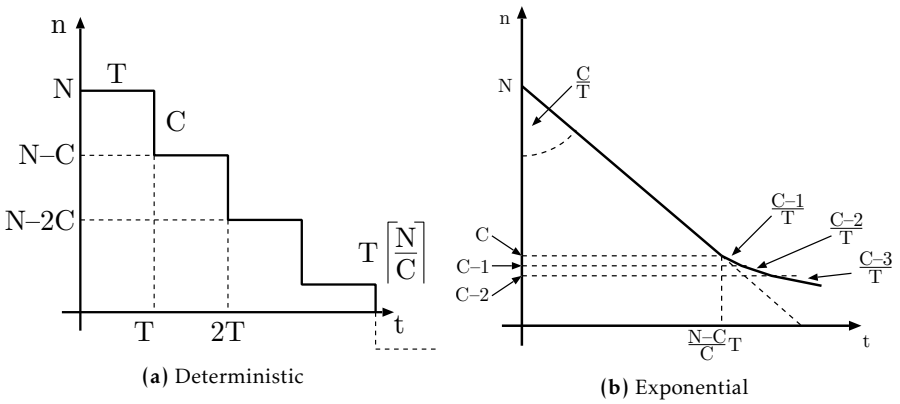
Following Barbierato et al. [21], the fluid evolution is defined as a function $\phi : \mathbb{R}^2 \to \mathbb{R}$, which defines how the fluid level of the corresponding place changes with time. In particular, $\phi(x,t)$ represents the fluid level reached by a place at time t, given that it starts with level x at time $t = 0$. This semantic is represented graphically by drawing a fluid arc that connects a fluid place (*Queue* in Figure 3.4) to a time-dependent fluid transition (*Exec* in Figure 3.4). In the model, two functions $\phi^{\mathrm{M}}(x,t)$ and $\phi^{\mathrm{R}}(x,t)$ are associated respectively to the map and reduce phases. These functions regulate the evolution of the fluid in the corresponding places, so that the fluid level x represents the average remaining number of tasks that still need to be executed in a phase.

Task duration distributions can be generally regarded as phase type or Erlang [9] parameterized according to the observed coefficient of variation (CV). Now, Erlang distributions have a CV in the range $[0,1]$: for this reason we approximate the probability distribution function (PDF) with an appropriate convex combination of the limiting cases with CVs equal to 0 or 1. Hence Section 3.4.2 derives the exact fluid evolution function under deterministic hypothesis, or CV 0, while Section 3.4.3 is about the exponential case, i.e., CV 1.

3.4.2 Deterministic Task Execution Time

Let us suppose that our MapReduce jobs are executed by C containers. Let us also assume that a phase is composed of N tasks and that each task requires a deterministic time T to be executed. Figure 3.5a represents the evolution of the number of remaining tasks as function of time, which in our fluid model corresponds to the fluid evolution function $\phi(N,t)$. At time $t = 0$, C out of N tasks are immediately assigned to all the containers. Since their duration is deterministic, all the C tasks will end at the same time T, leaving just $N - C$ tasks left to be executed in the system. Then the next batch of C tasks will start, and it will end at $2T$, leaving in the system $N - 2C$ jobs. The function $\phi_{\mathrm{d}}(x,t)$ can then be defined as follows:

$$\phi_{\mathrm{d}}(x,t) = \max\left(0, x - \left\lfloor \frac{t}{T} \right\rfloor C\right). \tag{3.1}$$



(a) Deterministic **(b)** Exponential

Figure 3.5 – Fluid evolution of task execution times

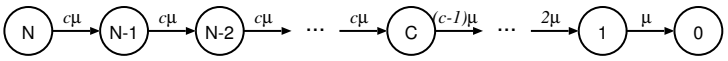


Figure 3.6 – MC for stages with exponential service times

In this scenario, the time $\tilde{t}_{\mathrm{d}}(N,C,T)$ required to run N tasks with deterministic running time T on C containers can be computed as:

$$\tilde{t}_{\mathrm{d}}(N,C,T) = T\left\lceil \frac{N}{C} \right\rceil. \tag{3.2}$$

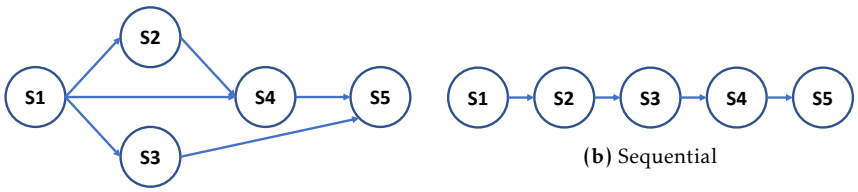
3.4.3 Exponential Task Execution Time

Let us now suppose that the task execution time is exponentially distributed, with average execution time T. Since the minimum of n exponential distributions of rate μ is exponentially distributed with rate $n\mu$, we can describe the evolution of the number of remaining jobs with the death-only birth-death process represented in Figure 3.6, where $\mu = \frac{1}{T}$. In particular, the average sojourn time in states N to C is $\frac{T}{C}$, while for the states $c \in \{C-1,\dots,1\}$ is $\frac{T}{c}$. This can lead us to the definition of $\phi_{\mathrm{e}}(x,t)$, as shown in Figure 3.5b: when $x \geq C$ the number of jobs decreases at rate $\frac{C}{T}$, then the decrease rate reduces at $\frac{\lceil x \rceil}{T}$ when $x < C$. This represents the fact that, when only fewer tasks than resources are available, the system cannot take full advantage of its computational capacity, instead only as many as $\lceil x \rceil$ can proceed in parallel. Overall we obtain:

$$\phi_{\mathrm{e}}(x,t) = \begin{cases} x - t\frac{C}{T}, & \text{if } t < \frac{N-C}{C}T \\ c - (t-t_c)\frac{c}{T}, & \text{if } t_c \leq t < t_{c-1} \end{cases}, \tag{3.3}$$

where t_c represents the time at which only c of the nodes are still in use, and it can be computed in a recursive way starting from $c = C$ down to $c = 1$ with the following formulas:

$$t_c = \begin{cases} \frac{N-C}{C}T, & \text{if } c = C \\ t_{c+1} + \frac{T}{c}, & \text{if } 1 \leq c < C \end{cases} \tag{3.4}$$



(a) Original

(b) Sequential

Figure 3.7 – DAG and order preserving sequential execution

In this scenario, the time $\tilde{t}_e(N, C, T)$ required to run N tasks with exponentially distributed running time with average T on C nodes can be computed as:

$$\tilde{t}_e(N, C, T) = T\left(\frac{N-C}{C} + \sum_{c=1}^{C} \frac{1}{c}\right). \tag{3.5}$$

3.4.4 General Task Execution Time

Typically, real data does not follow exactly neither deterministic nor exponential service time distributions, instead we can observe samples with diverse CVs. In order to take into account this variability and apply the proposed technique in the general case, our approach considers a convex combination of the previously presented limiting cases. The fluid evolution function becomes:

$$\phi(x, t) = \alpha\phi_d(x, t) + (1-\alpha)\phi_e(x, t), \tag{3.6}$$

which, in turn, yields:

$$\tilde{t}(N, C, T, \alpha) = \alpha\tilde{t}_d(N, C, T) + (1-\alpha)\tilde{t}_e(N, C, T). \tag{3.7}$$

Given the experimental data, we can profile each phase to extract the number of involved tasks and the average service time. Moreover, the α parameter can be obtained minimizing the absolute error between the measured phase duration and the predicted time $\tilde{t}$. For instance, a MapReduce fluid job profile consists of the following parameters: N^M, T^M, α^M, N^R, T^R, and α^R.

3.4.5 Spark DAG Stages Generalization

Spark applications and jobs are characterized by generic DAGs, as shown in Figure 3.7a. A generalized model to support Spark jobs should then use more fluid places to represent the various stages of the DAGs and exploit the discrete part of the model to properly share the available resources among the tasks that are ready to start. However, in many cases a DAG can be simplified as a sequence of stages. This result has been proven in scheduling theory for real systems in which the number of tasks N is greater than the available resources C [102]: a DAG execution can be approximated with a stage sequence respecting precedence constraints.

3.5 Models for Convolutional Neural Networks

Deep learning is gaining wide application in several domains, thanks to its effectiveness for a number of different tasks. Recently even Apache Spark is beginning to support both deep and transfer learning.[3] Within this section, however, we will refer to a framework specifically designed for CNNs, Caffe.[4] In terms of studied networks, the focus is on AlexNet [74], GoogLeNet [115], and VGG-16 [113], since they are quite well known for the good results obtained in the ImageNet Large Scale Visual Recognition Challenge[5] over time.

In this section, we present two methods to learn performance models for CNNs running on a single GPGPU. The main metrics under investigation are the forward time, relevant to quantify the time taken for classification when the trained network is deployed, and the gradient computation time, which on the other hand is important during the learning phase. In the following, we will generally mention forward and backward passes, referring to the direction in which information flows through the CNN, with either features incrementally processed for classification or partial derivatives that back propagate to reach learnable weights.

Our goal is to lead new users with limited previous experience from an initial test deployment to real scale applications. In order to meet the requirements of both scenarios, i.e., the generality needed in the preliminary design phase of a project on one side and the high accuracy expected when running in production on the other, we discuss two alternative methods to derive models that boast different trade-offs.

At first, we propose a gray box *per layer* approach where modeling is performed layer by layer and the only explanatory variable is computational complexity. This technique allows for a great deal of generality, since partial layer predictions are easy to sum and obtain an overall performance estimate for the full CNN, even if the specific network schema has never been considered as part of the training set. Due to this, the approach is preeminently interesting during the initial design stages, for instance to compare different alternative CNNs and deployments in terms of performance. What is more, designers can get a feeling of the resulting performance without ever needing to hit a cluster or the cloud for experiments, which is advantageous regarding both saved work hours and plain monetary savings. On the other hand, when a deployment is already available, data coming from the real system enables a different approach with its focus on precision rather than ease of generalization. Details about this approach can be found in Section 3.5.1.

This second scenario can be tackled with a black box approach, called *end to end* modeling, which focuses on a single CNN and learns the dependency of execution time on varying batch size and iterations number. The improved accuracy comes at the expense of a quite narrower focus centered around a particular network architecture and deployment, which prevents the application of the model in different situations. Moreover, this modeling technique requires a collection of historical data, thus entailing either an experimental campaign or the proper monitoring of previous runs in a production en-

[3] https://databricks.github.io/spark-deep-learning/site/index.html

[4] http://caffe.berkeleyvision.org

[5] http://www.image-net.org/challenges/LSVRC/

vironment. As we will demonstrate trough our empirical analyses, see Section 5.5.2, small scale experiments are enough to extrapolate to a larger scale range, which mitigates the cost of this approach. The technique used to obtain these end to end models is described in Section 3.5.2.

3.5.1 Per Layer Model

We first present a modeling technique aimed at achieving great generality to characterize the basic building blocks that compose CNNs. It allows for performance prediction even on network architectures never taken into account during the learning phase.

All CNNs comprise a number of layers belonging to a limited collection of basic categories. Building upon this observation, we propose to learn several linear regression models, in order to characterize common layer types. In this way, it is possible to estimate the performance of a wide range of CNNs even without previous experience with the specific structure, just by mixing and matching these low level layer models.

Since every CNN is specified in terms of its connectivity structure and a possibly extensive list of hyper-parameters, our approach is based on two basic assumptions, in order to make the problem easily tractable and improve model generality.

When working with GPUs, applications attain their best performance when they fully leverage the data parallel architecture, hence we expect CNN designers, as well as users, to tune networks accordingly. Such a consideration means that, mostly, the execution of different layers will not overlap, whence follows that layer running time predictions can just be summed to obtain an estimate of the overall execution time:

$$\hat{t}^{\text{CNN}} = i \sum_{l \in L} \hat{t}_l \tag{3.8}$$

where i is the total number of iterations.

The second aspect to take care of is the choice of features to feed into the regression models. A simplistic idea could be using all the various hyper-parameters as features, but this would make for a difficult to interpret and hardly generalizable formulation. On the other hand, we propose to summarize all the relevant characteristics in a single feature: layers computational complexity in terms of simple primitives available on GPGPUs. It is quite intuitive that such quantity is a good metric for the workload needed to complete a layer's computation, as performance will strongly depend on the bulk of algebraic transformations involved in the forward and backward passes through the CNN.

To exemplify the derivation of computational complexity from network hyper-parameters, in the following we present the detailed method for the namesake convolutional layer. Before delving deep into the algebra, we set forth the notation.

The convolutional part of CNNs operates on 3D tensors. With the notation $3 \text{ CH } 4 \times 5$ we represent a tensor with three channels stacked depth-wise, where each slice is a matrix with four rows and five columns. In formulas, C will be the number of channels, H the number of rows or height, W the number of columns or width. Filters are represented with their height and width

Table 3.1
Operations per Output Pixel

Layer	Forward	Backward
Conv	$H_f W_f C_{in} C_{out}$	$(2H_f W_f C_{in} + 1) C_{out}$
FC	$H_{in} W_{in} C_{in} C_{out}$	$2H_{in} W_{in} C_{in} C_{out}$
Loss	$4C_{out} - 1$	$C_{out} + 1$
Norm	$5C_{out} + C_n - 2$	$8C_{out} + C_n - 1$
Pool	$H_f W_f C_{out}$	$(H_f W_f + 1) C_{out}$
ReLU	$3C_{out}$	$4C_{out}$

only, since the general assumption with CNNs is that filters shift across the spatial dimensions, but always span all the input channels. The amount of zero padding on each side of the matrices is P, whilst the stride is S. We use subscripts to distinguish properties of the input, output, and filters, e.g., H_{in}, H_{out}, H_f. Cardinalities are used as a shorthand for index sets, as in $i \in H_{in}$.

Some layers just apply predetermined operations, possibly depending on hyper-parameters under users' control. In contrast, convolutional and fully connected layers have a set of learnable weights that evolve during the training phase via back propagation. The number of weights depends on their hyper-parameters. Each output channel is obtained by convolving a different filter with the input tensor, hence the count of learnable parameters is given by:

$$(H_f W_f C_{in} + 1) C_{out} \tag{3.9}$$

Convolution entails multiplying C_{in} ch $H_f \times W_f$ filters element-wise with input activations and producing as output the sum of all these partial products and an additive bias, hence there are C_{out} filter-bias pairs that contribute all the entries in the tensor plus one coefficient. Equation (3.9) applies to fully connected layers as well: they can be seen as a special case of convolutional ones, with the difference that their filters are as large as the input tensor.

The 3D tensors involved in CNNs contain all the partial values, called activations, obtained via the incremental transformations operated by filters. In practice, layers take an input tensor and apply a filter to its entries, thus yielding an output tensor with a possibly different layout. It is possible to compute output dimensions given layer hyper-parameters, specifically filter sides, padding around the edges, and stride:

$$H_{out} = \frac{H_{in} + 2P - H_f}{S} + 1 \tag{3.10}$$

where an analogous formula can be written for W as well. Following the dependencies among layers, one can easily determine the dimensions of each and every tensor in a CNN.

Tensor sizes are relevant because they appear in the formulas for computational complexity, since every output activation comes from one of the several applications of the filter to its inputs. Due to this reason, it is common to consider complexity per pixel, as the overall layer operations count is always directly proportional to $H_{out} W_{out}$. In particular, continuing our exam-

ple, convolutional layers can be formalized as the following expression for all $(i, j, k) \in H_{\text{out}} \times W_{\text{out}} \times C_{\text{out}}$:

$$y_{ijk} = b_k + \sum_{t \in H_{\text{f}}} \sum_{u \in W_{\text{f}}} \sum_{l \in C_{\text{in}}} w_{tulk} x_{\tilde{\imath}\tilde{\jmath}l} \tag{3.11}$$

where $\tilde{\imath} = \varphi(i, t)$ and $\tilde{\jmath} = \psi(j, u)$, while x and y are, respectively, input and output activations. φ and ψ associate output and filter indices to the input needed to convolve each activation and their specific functional forms depend on the CNN and its hyper-parameters, in particular padding and stride, but they do not affect the derivation of complexity. Overall, you multiply all the weights times the input activations and accumulate the products on the bias once per output channel, hence convolutional layers require $H_{\text{f}} W_{\text{f}} C_{\text{in}} C_{\text{out}}$ operations per output pixel.

Back propagation through a CNN consists in accumulating the contributions to partial derivatives of the loss function from all the layers, so as to analytically obtain the full gradient with respect to learnable weights. Exploiting the chain rule, at each layer it is possible to evaluate the exact derivative with respect to each parameter or input activation just by proper transformation of the ones relative to output activations. Specifically, recalling (3.11), we have:

$$\frac{\partial y_{ijk}}{\partial w_{tulk}} = x_{\tilde{\imath}\tilde{\jmath}l} \tag{3.12a}$$

$$\frac{\partial y_{ijk}}{\partial b_k} = 1 \tag{3.12b}$$

$$\frac{\partial y_{ijk}}{\partial x_{\tilde{\imath}\tilde{\jmath}l}} = w_{tulk} \tag{3.12c}$$

Due to parameter sharing, as filters shift across the input tensor, all the complete partial derivatives in these layers collect several contributions from outputs. Following a common practice, we adopt the light notation using deltas to represent partial derivatives of the loss function. With this convention, in order to obtain $\boldsymbol{\delta}^{(\mathbf{x})}$, for all $(i, j, l) \in H_{\text{in}} \times W_{\text{in}} \times C_{\text{in}}$ the formula reads:

$$\delta^{(\mathbf{x})}_{ijl} = \sum_{t \in H_{\text{f}}} \sum_{u \in W_{\text{f}}} \sum_{k \in C_{\text{out}}} w_{tulk} \delta^{(\mathbf{y})}_{\hat{\imath}\hat{\jmath}k} \tag{3.13}$$

where $\hat{\imath} = \varphi^{-1}(i, t)$ and $\hat{\jmath} = \psi^{-1}(j, u)$, where the inversion is intended on the spatial indices in the tensors, i.e., one inverts the one-variable functions of the form $\varphi(\cdot, t), \forall t \in H_{\text{f}}$. In (3.13) the partial derivatives with respect to output activations are propagated following backwards all the relevant filters, multiplied by the partial derivative of each output against inputs obtained in (3.12c), and summed so that every input activation collects contributions from all the filters. The evaluation of $\boldsymbol{\delta}^{(\mathbf{w})}$ similarly involves aggregating deltas from the output activations according to the filters' movement, whilst biases receive all these contributions once. Hence propagating deltas to biases requires one operation per channel, whilst doing so for parameters and inputs costs twice as much as the forward pass, because it fundamentally amounts to following the convolution backwards once for weights and once for activations. All in all, each output pixel requires $(2 H_{\text{f}} W_{\text{f}} C_{\text{in}} + 1) C_{\text{out}}$.

Table 3.2
CNN Characteristics, Batch Size 1

Network	Layers	Weights	Activations	Complexity
AlexNet	8	6.24E7	1.41E6	3.42E9
GoogLeNet	22	1.34E7	6.86E6	4.84E9
VGG-16	16	1.38E8	1.52E7	4.65E10

Table 3.1 shows formulas for all the kinds of layers, with the computational complexity per output pixel for both the forward and backward passes. Now, using these formulas it is possible to build a dataset where the operations count is associated with the measured execution times of both passes: given this data, we can build a series of models where computational complexity is the only explanatory variable. For every layer category and direction we learn, following the theory for linear regression, a model of the form:

$$t_l = \beta_{0l} + \beta_{1l} c_l + \varepsilon_l \tag{3.14}$$

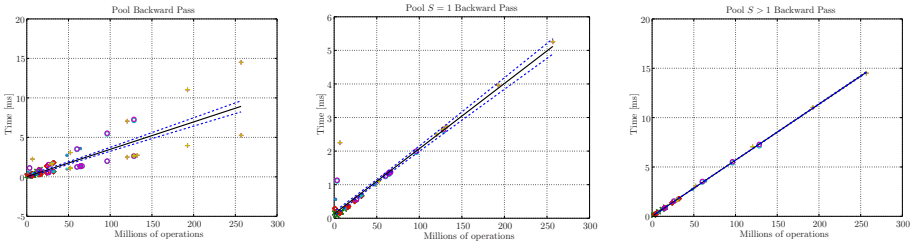
where t_l is the execution time, c_l the complexity, and $\varepsilon_l \sim \mathcal{N}\left(0, \sigma_l^2\right)$ random errors. With the estimated coefficients $\hat{\boldsymbol{\beta}}$ it is possible to predict both forward and back propagation time, then all the relevant contributions are added to obtain the time taken for one iteration. Multiplying by the overall number of iterations and summing the terms due to each layer, as in (3.8), yields a prediction for the full run. In particular, it is possible to predict both training and deployment execution times, depending on whether back propagation terms are included in the sum.

The choice of using only computational complexity as independent variable confers a lot of generality, allowing to learn a set of models on data coming from a limited selection of CNNs and to apply it nonetheless to different networks. Anyhow, the underlying hardware configuration obviously affects performance: not adding explicitly any contribution related to hardware in the formulation makes every trained model specific of the deployment where data is extracted. Practically this means that applying our method to a new GPGPU requires an initial experimental phase to obtain timing data relevant for the different accelerator.

Table 3.2 lists several interesting properties that characterize CNNs' performance, obtained with the approach described so far. For each of the studied networks, we list its number of layers, overall learnable weights, activations, and complexity at batch size 1. As the batch size increases, activations and complexity follow a direct proportionality. These quantities allow for several considerations on CNNs, for instance it is possible to assess what is the largest batch size that can fit in a GPU's memory based on the number of activations.

Since per layer models can be considered gray box approaches, they provide not only a performance prediction device, but also some interesting insight about the way CNNs work. Here we discuss some of the relevant observations made possible by the modeling technique.

To begin with, some layers, though different, can be merged into a single category without degrading the goodness of fit. Convolutional and fully con-



(a) All strides **(b)** $S = 1$ **(c)** $S > 1$

Figure 3.8 – GoogLeNet pooling layers, backward pass

Table 3.3
Operation Count and Layer Time Breakdown, GoogLeNet

Category	c_l^{fw} [%]	t_l^{fw} [%]	c_l^{bw} [%]	t_l^{bw} [%]
Conv/FC	98.33	69.80	98.94	67.67
Norm	0.25	2.62	0.20	2.09
Pool	0.82	16.35	0.46	22.18
ReLU/Drop	0.60	11.23	0.40	8.07

nected layers are an intuitive example of this behavior, as the former are just a generalization of the latter. A less obvious similarity was found between rectified linear units and dropout layers. The former are the most common expressions that provide a threshold effect on activations, thus introducing the nonlinearities that offer CNNs' representational power, whilst dropout is a technique adopted to improve classification accuracy [114]. Likely the performance of both can be characterized with just one pair of models due to the fact that both need to loop through activations and act based on a point-wise condition.

Even if some layers can be easily modeled as part of a wider category, we nonetheless observed that pooling, in contrast, requires the adoption of multiple categories based on stride. In order to highlight this aspect, in the following we report some preliminary results on GoogLeNet from the GPGPU deployment described in detail in Chapter 5. The Caffe framework allows for running a CNN while profiling each layer, thus providing their average performance. In these plots, different markers represent data coming from timing runs with different batch sizes, the solid line is the model of layer execution times against complexity obtained via linear regression, and the dotted lines are the boundaries of the 95 % confidence interval around the predicted mean layer time. Figure 3.8a clearly shows two different behaviors for layers operating at $S = 1$, which mostly lay on the smaller slope line, and for the ones with $S > 1$, aligned on greater times. As you can see, it is not possible to obtain a satisfactory model for pooling layers irrespective of stride. On the other hand, Figures 3.8b and 3.8c prove that separate linear models can effectively fit the measurements. Our interpretation of this phenomenon is that the change in stride affects GPGPUs' memory access patterns, causing a degradation in performance.

As part of our investigation of CNNs' performance, we initially focused on comparing the breakdown of operation counts and layer execution times.

Table 3.4
Linear Regression Models, NVIDIA Quadro M6000

Category	$\hat{\beta}_0^{\text{fw}}$ [ms]	$\hat{\beta}_1^{\text{fw}} \left[\frac{\text{ms}}{\text{op}}\right]$	$\hat{\beta}_0^{\text{bw}}$ [ms]	$\hat{\beta}_1^{\text{bw}} \left[\frac{\text{ms}}{\text{op}}\right]$
Conv/FC	1.83E−1	3.43E−10	3.15E−1	3.65E−10
Norm	1.64E−2	7.11E−9	1.01E−1	6.87E−9
Pool $S = 1$	2.23E−2	1.27E−8	1.52E−1	1.93E−8
Pool $S > 1$	1.44E−2	1.45E−8	6.11E−3	5.67E−8
ReLU/Drop	8.91E−3	1.17E−8	1.18E−2	1.33E−8

Table 3.3 summarizes this analysis on data coming from our experimental deployment, in particular we present the results for GoogLeNet only because all the three considered CNNs showed an analogous behavior. Both for the forward and backward pass, convolutional layers account for more than 98 % of the overall computational complexity, yet the time taken for their processing remains below 70 % of the total. This pattern highlights how GPGPUs are actually a good tool for CNNs, as they optimize precisely for the most common kind of performed operations. Along the same lines, these breakdowns can be useful in designing special purpose devices, such as the recent NVIDIA Volta V100, which boasts *tensor cores* specifically devised for the matrix operations that make up most part of convolutional layers computation.

Table 3.4 reports the linear regression coefficients by category, both for the forward pass and back propagation. Following the notation established in equation (3.14), this table shows the estimated intercepts and slopes. These models were obtained on data for an NVIDIA Quadro M6000 GPU. Convolutional and fully connected layers achieve the best marginal efficiency, followed by normalization and, on a similar level, all the other categories. Our statement is motivated by the lower slopes needed to fit the data: this means that, when the complexity increases by a fixed amount, the convolution/fully connected category suffers a relatively smaller impact in comparison to other layer kinds. Such effect corroborates the observations previously discussed with respect to the operations and time breakdowns.

3.5.2 End to End Model

The per layer model presented in Section 3.5.1 offers wide generality, yet exploiting more run time measurements obtained in a specific scenario can lead to more accurate predictions for that particular case. Users who already own a working deployment and settle on one or several specific CNNs are likely eager to trade off some generality for lower prediction errors. The current section shows an alternative model that allows for the mentioned compromise.

The basic idea is to extract from historical data, particularly logs of previous runs or traces collected by a monitoring platform, the execution time of the network in its entirety, so as to build a dataset associating these timings to batch sizes and number of iterations. Then it is possible to apply linear regression to a sample in order to obtain a model specialized for the particular CNN and deployment under consideration, but capable of predicting performance with high accuracy.

Deep learning practice usually involves several alternating phases of CNN

training and testing. The former iteratively feeds the network with labeled image batches, so that its parameters can change following the direction of the back propagated gradient, whilst the latter evaluates the CNN's evolving quality in terms of more human readable metrics, rather than the loss function used for training, but without contributing to the learning of weights and biases. For example, generally training is performed minimizing a loss function that may be SVM-like or based on cross entropy, but the stopping criterion is likely expressed in terms of classification accuracy or F-score, for unbalanced datasets. Since training involves back propagation, but testing does not, it is necessary to characterize two different models, one per phase, in order to correctly catch relevant behaviors.

An interesting aspect to consider is the choice of features for the design matrix. When the use case is more focused on working with fixed batch size or, conversely, fixed iteration number, then it is straightforward to use only the varying axis as explanatory variable. In both ways first degree polynomials yield an accurate representation of the dependency of execution time on batch size or iterations. The same does not apply to models learned against a dataset with both batch size and number of iterations that vary. However, since the results in the single variable case corroborate separately affine relations (i.e., in the form $ax + b$) of the execution time with either variable, the following Theorem 3.5.1 guarantees that the only higher degree term to consider is the quadratic interaction.

Theorem 3.5.1. *Let $F : \mathbb{R}^2 \to \mathbb{R}$. F is affine in x for all $y \in \mathbb{R}$ and, symmetrically, is affine in y for all $x \in \mathbb{R}$. Then, F is a second degree polynomial of the form:*

$$F(x, y) = axy + bx + cy + d$$

Proof. Due to affinity, for any $x \in \mathbb{R}$ we can write:

$$F(x, y) = f(x)y + g(x)$$

Again, affinity guarantees that for all $y \in \mathbb{R}$ the pure second order partial derivative with respect to x is null:

$$F_{xx}(x, y) = f''(x)y + g''(x) = 0$$

By equating the coefficients, it follows that $f'' = 0$ and $g'' = 0$, so f and g are themselves affine in x, whence the thesis. □

Thanks to Theorem 3.5.1 and knowing that, fixed every other variable, execution time shows an affine dependency on either batch size or iterations number, it follows that the overall dependency when both quantities vary can be expressed as a quadratic polynomial where the only second degree term is the batch-iterations product.

The same considerations discussed in Section 3.5.1 about the limitations in applying the learned models to different GPGPUs hold for this approach as well. Particularly, in this case they entail that each pair of training/classification performance models are specialized for a single CNN deployed on exactly one hardware configuration. However, this second method is devised for use cases where generality is less relevant, thus making the constraint less problematic in practice.

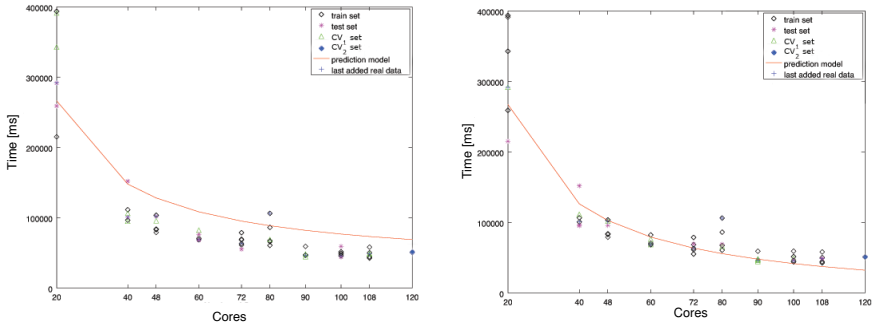
3.6 Hybrid Machine Learning

Often big data applications users need to know beforehand how long their job execution will take using different configurations for the cloud infrastructure they want to rent. In other words, they want to determine how jobs execution time changes when the available resources (in terms of, e.g., VMs type and their number) changes. However, running experiments in real cloud environments is generally expensive and time consuming. So, exploiting a reasonably accurate model for performance evaluation and prediction of cloud applications is of great importance.

Our work starts from the bootstrapped hybrid performance modeling proposed by Didona and Romano [45], which is a combined AM/ML modeling approach that brings the strengths of AM methods to compensate the weaknesses of ML techniques, and vice versa. On the one hand, hybrid approaches use AMs, which rely on a priori knowledge of the internals of the target system, hence are a white box approach. On the other hand, ML infers the relationships that map application and system characteristics onto the target performance indicators through statistical models, without requiring the knowledge of internal system details, thus a black box approach. While white box techniques offer good *extrapolation* capabilities, i.e., they are able to predict values in regions of the parameters space not sufficiently explored, black box ones provide good *interpolation* capabilities, i.e., to predict values in areas of the features space that have been sufficiently observed during the training phase. So, utilizing bootstrapped hybrid techniques allows for achieving the best of both worlds. In particular, hybrid methods provide: (i) more robust performance predictors that require a small training phase in order to instantiate a performance model, (ii) good predictive performance because of extrapolation capabilities (both borrowed from AMs), (iii) the ability to progressively enhance the accuracy of the performance predictor as new data samples from the operational system are gathered, and (iv) good interpolation capabilities (both borrowed from ML).

Didona and Romano's approach exploits an early AM to generate an initial set of synthetic data points, which are then fed into a ML model to predict the application performance. Then the KB of synthetic data is updated by real samples over time following either the merge or replacement strategies to achieve a good accuracy. According to the former strategy, real samples are collected from the real operational system and added to the synthetic set, while in the replacement case, the nearest neighbor is removed and a new real sample is incorporated into the initial KB. Consequently, the ML model is also updated and trained according to the new KB. A critical consideration during the aggregation of real data for the training process is that a limited number of configurations can be analyzed at design time for cloud big data applications. Moreover, in some cases real data samples might be noisy (for instance, when big data applications run on shared cloud infrastructure) and should be consumed conservatively.

Regarding our work, an iterative procedure was adopted for merging real data from the operational system into the KB. Since in our case both synthetic and real data come from a limited size configuration set, if we choose the replace strategy in the first iteration of our incremental and iterative process of model selection and training new real data points evict the synthetic ones



(a) Noisy data is added

(b) After reshuffling the sample

Figure 3.9 – Model oscillation controlled by the hybrid algorithm

of the same configuration. Then, in each subsequent iteration, new real data points evict the old real ones. Therefore, the output model of each iteration will be trained on the latest available set of added (possibly noisy) data points and the accuracy will not necessarily improve over time, instead it will be subject to oscillations. Hence, a model obtained though the replacement strategy cannot be used in action, since relying on few possibly noisy data samples often generates very inaccurate performance predictions.

On the other hand, when the merge strategy is implemented the results are more accurate and dependable than the ones based on the replace strategy, as soon as the number of iterations becomes large enough. However, since both the size of the configuration set and the number of iterations are rather small in our scenarios, naively applying Didona and Romano's approach, the prediction curve oscillates occasionally during successive iterations and the error may become large even in the last iteration, which can produce unacceptable prediction outputs.

As an example, Figure 3.9a shows a prediction model which is generated after adding some new noisy data into the KB. The x-axis represents the number of cores for different configurations and the y-axis shows the response time of MapReduce jobs in milliseconds. Based on how examples arrange among training, test, and validation sets, it is possible that noisy measurements exert a strong leverage effect and make the learned model quite inaccurate: Figure 3.9a shows precisely a model driven quite off the bulk of execution times by a few outliers, with a MAPE that ends up being 72 %. The algorithm proposed in the following sections addresses this issue by reshuffling the KB whenever the MAPE on a validation set goes above a given thresholds, with the resulting model shown in Figure 3.9b, where the MAPE becomes 19.5 %.

The goal of this section is to describe the hybrid ML models we developed to predict the performance of Hadoop MapReduce, Tez, and Spark applications running on clusters managed by the YARN Capacity Scheduler. First of all, Section 3.6.1 discusses two alternative AMs for producing artificial data samples for the hybrid approach. Then, Section 3.6.2 details possible ML techniques and the features to include in the dataset. In the end, Section 3.6.3 lists all the steps of the proposed algorithm.

3.6.1 Analytical Models

Hybrid ML is a general method to augment and complement analytical modeling techniques and ML, without constraining either in any way. Due to this consideration, we compared various AMs and ML formulations. The first AM we considered is the QN discussed in Section 3.2, which is used for modeling, for the sake of simplicity, a MapReduce job execution in a cluster of computing servers.

As a second model, we consider a first principle formula that approximates DAGs execution as a sequence of stages respecting precedence constraints (see Figure 3.7). As discussed in Section 3.4, such approximation has been proven to be accurate in scheduling theory for systems in which the number of tasks is greater than the available resources. This happens very frequently in practice especially for Spark jobs, which are characterized by thousands of tasks [99] that take a few milliseconds each. A DAG execution time is estimated as:

$$D = \sum_{i \in S} \left\lceil \frac{n_i}{c} \right\rceil t_i, \tag{3.15}$$

where S is the set of all the stages in the job, n_i is the number of tasks and t_i the average execution time of each task associated with stage i, while c is the number of cores available for execution.

The term $\left\lceil \frac{n_i}{c} \right\rceil$ yields the number of waves required to complete all the tasks at stage i. During the first $\left\lceil \frac{n_i}{c} \right\rceil - 1$ waves, tasks statistically keep all the c cores busy, while during the very last wave the remaining tasks complete the execution of stage i, possibly without saturating the c cores.

Equation (3.15) is pretty similar to (3.2). Specifically, the overall service demand, D, is approximated summing up several terms like (3.2), one per stage. In practice we are giving a first order approximation of the execution time under the assumption that task durations are deterministic, without considering possible improvements due to overlaps between subsequent stages for the sake of simplicity.

3.6.2 Machine Learning Model

In this work, machine learning is used to regress execution time of Hadoop and Spark applications in a cloud cluster. Different techniques have been investigated (for additional details see Ataie et al. [15]) including linear regression, Gaussian SVR, polynomial SVR with degree ranging between 2 and 6, and linear SVR. As feature set, we started from a diverse collection of features including the number of tasks in a map/reduce phase, or Spark stage, average and maximum values of tasks execution time, average and maximum values of shuffling time, dataset size, and the number of available cores. The set of relevant features has been obtained by considering the analytical bounds for MapReduce clusters proposed in [90, 122].

Our initial experiments showed that Gaussian SVR and polynomial SVR do not predict accurately and the errors they produce are usually large. On the other hand, we found that the best results were most often achieved by linear regression and linear SVR. However, the linear regression model was unstable in frequent cases when linearly dependent features existed. Filtering

linear dependent features required ad hoc analysis and resulted to be query specific. For this reason, we preferred to adopt SVR, which is also notoriously more robust to noisy data. Instead of considering the number of cores as a feature, we considered $\frac{1}{c}$, building on the approximate formula (3.15).

In our work, we perform a model selection process. A four-way data splitting method has been applied for identifying the best ML candidate model. The available operational data samples in the KB are partitioned into four disjoint sets: training and test set, as well as two cross validation sets, V_1 and V_2. Moreover, since our goal is to identify models able to achieve generalization, the operational data obtained with the largest configurations are included only in V_2, as this has been demonstrated a relevant choice for the thresholds optimization step (see the next section and Section 5.4.1). While the training set is used for training different alternative models, the V_1 set is exploited for SVR model selection, while V_2 is used as stopping criterion and to determine the best values for the thresholds of our iterative algorithm. The test set is used for evaluating the accuracy of the selected model [13] and restricted to include some specific configurations in order to test interpolation or generalization capabilities.

3.6.3 Hybrid Algorithm

The pseudo-code of our proposed hybrid algorithm is shown in Algorithm 3.6.1. A synthetic data set, used to form an initial KB, is generated at line 1 based on one of the AMs discussed in Section 3.6.1 (i.e., approximate formula or simulation). The KB is then used to select and train an initial ML model at line 2. Since the real data samples are noisy, we need to avoid the dependency on this data as much as possible. So, an iterative procedure is adopted for merging real data from the operational system into the KB, which is implemented at lines 3–14. Adding a new configuration in the KB means that many operational data points are included and subsequently split into the training, test, and cross validation sets. The operational data for all available configurations is gathered and then merged into the KB at lines 4–7. Then the updated KB is shuffled and partitioned at lines 9 and 10. This shuffling is motivated as an attempt to arrange the samples in a way that noisy data has the least impact on the final model accuracy. Using these sets, line 11 is dedicated to the selection of an ML model between alternatives and its final training. Then some error metrics are measured at line 12.

At lines 13 and 14, two conditions are checked. Both conditions consider the MAPEs on the training set, $\varepsilon_{\text{train}}$, and on the second cross validation set, ε_{V_2}, to check whether or not they are smaller than specific thresholds, namely, τ^{in} and τ^{out}. The error on the training set determines if the model fits well its input data samples. So, if this error is small enough, the model will avoid underfitting, or high bias. On the other hand, the error on the V_2 set quantifies the model's generalization capability. So, if this error is sufficiently small, the model will avoid overfitting, or high variance.

If the values of errors for the inner condition are not small enough and the maximum number of inner iterations is not reached, the algorithm jumps to line 8 to reshuffle and split the KB into train, V_1, and V_2 sets and choose a different model. On the other side, the outer condition prohibits the emission of a weak model. As it will be demonstrated in the following, if a good value is

Algorithm 3.6.1 Hybrid algorithm

1: create a KB using synthetic data generated from AM
2: select and train an initial ML model
3: **repeat**
4: **for all** available configurations **do**
5: gather new data from operational system
6: **end for**
7: merge new data into KB using specified weights
8: **repeat** $k \leftarrow 1, 2, \ldots$
9: shuffle KB
10: partition KB into train, V_1, and V_2 sets
11: select and train new ML model
12: evaluate $\varepsilon_{\text{train}}$ and ε_{V_2}
13: **until** $\left(\varepsilon_{\text{train}} < \tau^{\text{in}} \wedge \varepsilon_{V_2} < \tau^{\text{in}}\right) \vee k = k^{\max}$
14: **until** $\left(\varepsilon_{\text{train}} < \tau^{\text{out}} \wedge \varepsilon_{V_2} < \tau^{\text{out}}\right) \vee$ no additional data is available

chosen for the thresholds, the first condition prevents the oscillation problem discussed in Section 3.6 and the second one stops the procedure as soon as a good model is obtained.

On the other hand, if the value of errors for the outer condition is not small enough, the algorithm jumps to line 3 to start another iteration. When both errors are smaller than τ^{out} or no new data from the operational system is available, the algorithm stops. If the errors are sufficiently small, the current model seems to be good enough for performance prediction and the iterative process can be stopped to avoid consuming more operational data for the matter of time and cost of real experiments. On the other hand, if no new data is available, the algorithm stops and outputs the last ML model.

3.7 Discussion

All the performance models presented so far can be exploited, based on their peculiarities, during the various phases of the optimization procedures described in Chapter 4. In particular, ML and hybrid techniques yield closed form algebraic equations, thus making them most appropriate for mathematical programming formulations, where such equations can be readily used as constraints and derived to apply Karush-Kuhn-Tucker (KKT) conditions. At the same time, these methods tend to be less accurate than simulation-based models and, most importantly, cannot be reliably used for prediction in regions of the design space not previously explored via experiments. On the other hand, despite requiring seconds or even minutes to run each simulation, the performance models devised via QNs or SWNs attain a good average accuracy and are less sensitive to the range where their evaluation is performed. QNs boast a 14.13 % relative error, whilst SWNs 9.08 %, as reported in Chapter 5.

Overall, the above presented models are instrumental to provide an answer to research question 1. Among them, some rely on ML, hence they require a preliminary experimental campaign in exchange for the fastest pre-

diction times, but they also suffer the defect of being unreliable outside of the training domain. Instead the models adopting QNs or SWNs, as already highlighted, are more robust with respect to the evaluated working point, yet trading off longer simulation times to achieve such guarantee.

In the following, according to the above mentioned considerations, ML models will be exploited to provide algebraic formulas for system performance, whereas other simulation-based formalisms are adopted to heuristically explore the state space starting from an informed initial guess. This approach allows to obtain as final outcome an accurate and reliable prediction, yet without wasting time in a never ending search, as it would be the case if beginning with a random configuration.

Personal Publications

The models discussed in this chapter have been published in the following research papers. Sections 3.2 and 3.3 refer to Ardagna et al. [9], a work submitted to, and awarded as best paper of, the 16th International Conference on Algorithms and Architectures for Parallel Processing. The material in Section 3.4 was first presented at the InfQ workshop [53], associated with ValueTools. This work was later extended and published in ACM Performance Evaluation Review as Gianniti et al. [52]. The hybrid techniques elaborated in Section 3.6 were presented in the MICAS workshop [15], associated with SYNASC.

CHAPTER 4

Optimization Models

The following chapter overviews and details the optimization techniques that form the core of this dissertation. Resource allocation, admission control, and other related problems require performance models capable of providing predictions with fair confidence, possibly with additional constraints on the overall time taken by the optimization process when this is aimed at run time application. This is the reason why Chapter 3 discusses several alternatives and highlights their differences in terms of accuracy and evaluation times. Here we will present a number of variants of a mathematical programming formulation for resource allocation of big data applications, underlining the trade-offs enabled by the alternative prediction techniques. At the same time we will explore a spectrum of scenarios, from design time considerations, such as capacity planning, to run time problems of cluster management.

Design time issues were addressed in the context of the European research project DICE H2020, whilst the run time part was developed as part of the Europe-Brazil collaboration project EUBra-BIGSEA H2020.

This chapter is so organized. Section 4.1 describes the architecture of the design time tool, D-SPACE4Cloud, and Section 4.2 presents the general assumptions of our problem statement. Later on, Section 4.3 discusses the optimization model devised for solving the design time problem. Then Section 4.4 details the architecture of the run time middleware developed within EUBra-BIGSEA, while Section 4.5 talks about the problem it solves. In the end, we discuss the main achievements in Section 4.6, alongside their relevance to research question 2.

4.1 D-SPACE4Cloud Design Time Architecture

The tool we present and discuss in this section, namely D-SPACE4Cloud, has been developed within the DICE[1] H2020 European research project [27]. The project aims at filling gaps in model driven engineering with regard to the

[1] Developing Data-Intensive Cloud Applications with Iterative Quality Enhancement

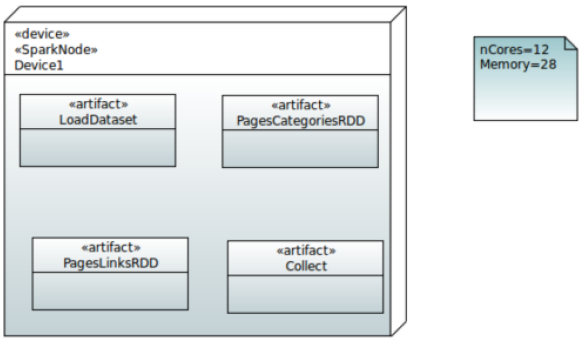


Figure 4.1 – Example DDSM

development of DIAs in cloud environments, embracing the DevOps [14] culture. DICE is committed to developing an integrated ecosystem of tools and methodologies intended to streamline the DIA development process through an iterative and quality-aware approach: design, simulation, verification, optimization, deployment, and refinement. DICE primarily proposes a data-aware UML profile that provides designers with the means necessary to model the dynamic and static characteristics of the data to be processed as well as their impact on the performance of an application's components. In addition, the project develops an IDE capable of supporting the managers, developers, and operators in quality related decisions. The IDE enforces the iterative design refinement approach through a toolchain of both design and run time tools. The former cover simulation, verification, and optimization of deployment, whereas the latter encompass deployment, testing, and feedback analysis of monitoring data.

At the core of DICE project's outcomes there are three main kinds of profiles for DIAs: DICE Platform Independent Models (DPIMs), DICE Platform and Technology Specific Models (DTSMs), and DICE Platform, Technology, and Deployment Specific Models (DDSMs). Since the optimization tool needs to interact with performance models for a particular technology, DPIMs are too abstract. On the other hand, DTSMs provide information about the logical components of an application and what framework or technology is used to realize them, whilst DDSMs add even more detail about the specific choice of underlying hardware, either physical or cloud resources. Given an initial guess as DTSM, the optimization tool can compare several alternative DDSMs (see an example in Figure 4.1) to highlight the most promising, also determining the cheapest configuration that meets QoS constraints. Moreover, after actually deploying the DIA, it is possible to refine the models by parsing application logs.

D-SPACE4Cloud is the DIA deployment optimization tool integrated in the DICE IDE. The tool serves the purpose of optimizing the deployment costs for one or more DIAs with performance guarantees. In a nutshell, within the quality aware development process envisioned by DICE, a DIA is associated with QoS requirements expressed in form of a maximum execution time, or deadline, and concurrency level, i.e., a number of users executing the same

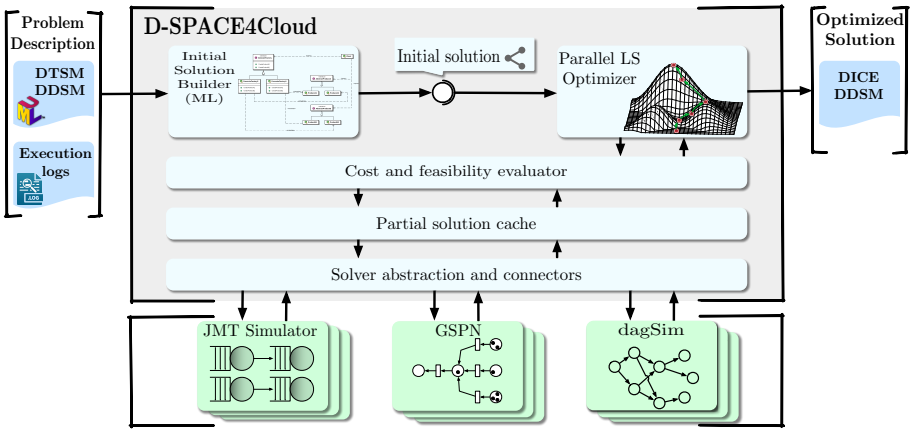


Figure 4.2 – D-SPACE4Cloud's architecture

application, with a certain think time, on a given system. D-SPACE4Cloud addresses and solves the capacity planning problem consisting in the identification of a minimum cost cluster, both for public and private clouds, to support the concurrent execution of several DIAs and their timely completion. To this end, the tool implements a design time exploration process able to consider multiple target VM candidates, possibly also across different cloud providers.

D-SPACE4Cloud supports the deployment optimization in the two distinct scenarios of public and a private cloud environments. The public cloud is mainly characterized by the fact that cluster resources (i.e., VMs) can be considered practically infinite for any common purpose. In this scenario, the concurrency level is not a problem and the tool focuses on selecting the most cost-effective VM type and number of replicas for each application. In the private cloud scenario, however, the cluster is provisioned on premises, the available resources are generally limited, and the resource allocation plan has to contemplate the possibility to exhaust the computational capacity before being able to provision a cluster that satisfies the QoS constraints. In such a situation, the tool can, if required, alter the underlying problem adding an admission control mechanism, which enables job rejection in the optimization process. In this dissertation we only discuss the public cloud scenario of design time problems, since the private cloud case boils down to a small extension exploiting well known mathematical programming formulations, namely, multidimensional knapsack and bin packing. Details can be found in Ardagna et al. [12]. On the other hand, Section 4.5 is more focused on the solution we developed for private cloud cases at run time within the frame of the EUBra-BIGSEA project.

Figure 4.2 depicts the main elements of D-SPACE4Cloud's architecture. Our tool is a distributed software system designed to exploit multi-core and multi-host architectures to work at a high degree of parallelism. In particular, it features a presentation layer (integrated in the DICE IDE) devoted to handle the interactions with users and the other components of the toolchain, an optimization service (colored gray), which transforms the inputs into suitable performance models [9] and implements the optimization strategy, and a

horizontally scalable assessment service (colored green in the picture), which abstracts the performance evaluation from the particular solver used under the hood. Currently, D-SPACE4Cloud supports a QN simulator (JMT [24]), a SPN simulator (GreatSPN [16]), and the EUBra-BIGSEA DES (dagSim [3]).

D-SPACE4Cloud takes in input:

1. A UML description of the applications sharing the cluster (see Artac et al. [14] for additional details on DICE UML models). In this context, DIAs are specified via DTSMs. Moreover, under specific circumstances, execution logs, for instance the ones obtained executing the applications in a pre-production environment, can replace DTSMs as input.

2. A partially specified deployment model for each application. The deployment model must be specified in DDSM format. This model is used as a template to fill and return in output.

3. A description of the execution environment, with a list of candidate providers and VM types along with VM performance profiles. These pieces of information are used to generate suitable performance models.

4. The list of QoS constraints, that is the concurrency level and deadline for each DIA, respectively.

The optimization service is the centerpiece of the tool. It primarily parses the inputs, stores the relevant information using a more manageable and compact format, then calculates an initial solution for the problem (via the *Initial Solution Builder*) and improves it via a simulation-optimization algorithm (implemented by the *Parallel Local Search Optimizer*).

The initial solution is generated by solving a mixed integer nonlinear programming (MINLP) formulation, whose perhaps most interesting feature is that some of its constraints have been modeled by applying ML techniques to the problem of estimating the execution time of DIAs. Different techniques have been investigated [15, 106], including linear regression, as well as Gaussian, polynomial, and linear SVR. The linear SVR was selected as it proved to be both accurate and robust to noisy data. More details are available in Section 4.3.

It must be highlighted, at this point, that the quality of the initial solution can still be improved, mainly because the MINLP relies on an approximate performance model obtained via ML. If such techniques' strong suit is regression within the range explored during the training phase, there is no guarantee on their accuracy when generalizing to other regions of the state space. Since at design time it is quite unlikely that users have extensive datasets for profiling, we exploit simulation-based performance models, which achieve a good accuracy without depending too much on the particular configuration under investigation. The difference in accuracy might cause the need to adjust the solution with a local search; however, since the simulation process is time consuming, the space of possible cluster configurations has to be explored in the most efficient way, avoiding the evaluation of unpromising configurations. The Optimizer component carries out this task, implementing a simulation-optimization technique to minimize the number of resource

replicas (i.e., VMs) for each application class. This procedure is applied independently, and in parallel, on all the application classes and terminates when a further reduction in the number of replicas would lead to an infeasible solution.

As soon as all the classes reach convergence, D-SPACE4Cloud leverages the optimized solution (selected provider, type and number of VMs per application) to update the DDSMs and provides them in output. Such models, in turn, can be converted into TOSCA blueprints and used to deploy the cluster exploiting another tool, DICER [14], that is part of the DICE toolchain.

4.2 Problem Statement

In this section we aim at introducing some important details on the problems addressed in this dissertation. We envision the scenario wherein a business venture needs to set up a cluster to carry out efficiently a set of interactive DIAs. A cluster featuring the YARN Capacity Scheduler and running on a public cloud IaaS is considered a fitting technological solution for the requirements of the company. In particular, the cluster has to support the parallel execution of DIAs in the form of Hadoop jobs and Hive/Pig/Spark-SQL queries or general Spark applications. Different classes $\mathcal{A} = \{i \,|\, i = 1, \ldots, n\}$ gather the applications that exhibit a similar behavior and share performance requirements. The cluster composition and size, in terms of type and number of VMs, must be decided in such a way that, for every application class i, h_i jobs are guaranteed to execute concurrently and complete before a prearranged deadline D_i. Moreover, YARN is configured in a way that all available cores can be dynamically assigned for task execution. Finally, in order to limit the risk of data corruption, and according to the practices suggested by major cloud vendors, the datasets reside on a cloud storage service accessible in quasi-constant time.

In general, IaaS providers feature a large catalog of VM configurations that differ in features (CPU speed, number of cores, available memory, etc.) and cost. Making the right design decision implies a remarkable endeavor that can be repaid by important savings throughout the cluster life cycle. Let us index with j the VM types available across, possibly, different cloud providers and let $\mathcal{V} = \{j \,|\, j = 1, \ldots, m\}$. We denote by τ_i the VM type used to support DIAs of class i and with ν_i the total number of VMs allocated to class i.

In this scenario, we consider a pricing model derived from Amazon EC2.[2] The provider offers: i) *reserved* VMs, for which it adopts a one time payment policy that grants access to a certain number of them at a discounted rate for the contract duration; ii) *on demand* VMs, which can be rented by the hour according to current needs; and iii) *spot* VMs, created out of the unused data center capacity. For such instances customers bid and compete, yielding very competitive hourly fees at the expense of reduced guarantees on their reliability.[3] In order to obtain the most cost-effective configuration, we rely on

[2] `https://aws.amazon.com/ec2/pricing/`

[3] The bidding mechanism was in place at the time of writing. Currently the spot fee is fixed by Amazon, but it remains the cheapest alternative due to the loose guarantees on reliability: the provider can terminate these instances on a two-minute notice, in order to reclaim data center capacity. However, the following formulation remains valid, since it does not consider fine-grained fluctuations of the spot price.

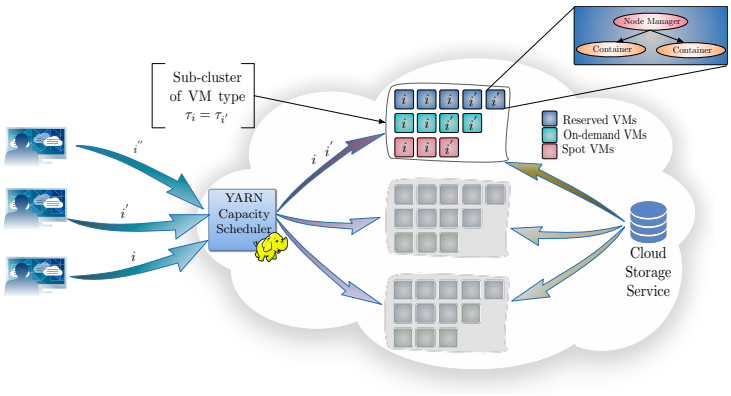


Figure 4.3 – Reference system

reserved VMs (denoting with r_i their number) to satisfy the core of computational needs and complement them with on demand (d_i) and spot (s_i) instances, hence $\nu_i = r_i + d_i + s_i$. Let ρ_{τ_i}, δ_{τ_i}, σ_{τ_i} be the unit costs for VMs of type τ_i, respectively, reserved, on demand, and spot. Overall, the cluster hourly renting out costs can be calculated as follows:

$$\sum_{i \in \mathcal{A}} \left(\rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \right). \tag{4.1}$$

As the reliability of spot VMs is susceptible to market fluctuations, to keep a high QoS level the number of spot VMs is bounded not to be greater than a fraction η_i of ν_i for each class i. In addition, reserved VMs must comply with the long term contract signed with the cloud provider and cannot exceed the prearranged allotments R_{ij}, where every class may have a separate pool of reserved VMs of any type at their disposal. It is worth noting that this cost model is general enough to remain valid, zeroing the value of certain parameters, even in those cases where the considered cloud does not feature reserved or spot instances. Specifically, if the IaaS provider does not offer reserved VMs it is enough to set $R_{ij} = 0$, while spot instances can be disabled with $\eta_i = 0$.

Reducing the operating costs of the cluster by using efficiently the virtual resources in lease is in the interest of the company. This translates into a resource provisioning problem where the renting out costs must be minimized subject to the fulfillment of QoS requirements, namely a per class concurrency level h_i given certain deadlines D_i. In the following we assume that the system supports h_i users for each class and that users work interactively with the system and run another job after a think time exponentially distributed with mean Z_i, i.e., the system is represented as a closed model [78].

In order to rigorously model and solve this problem, it is crucial to predict with fair confidence the execution times of each application class under different conditions: level of concurrency, size and composition of the cluster. The execution time can generically be expressed as:

$$T_i = \mathcal{T}_i(c_i, h_i, Z_i, \tau_i), \quad \forall i \in \mathcal{A}, \tag{4.2}$$

where c_i is the total number of cores devoted to application i. The proportionality between number of cores and VMs is established by the parameters Γ_j, which quantify how many CPUs are hosted in each instance of type j, that is $c_i = \Gamma_{\tau_i} \nu_i$. What is worthwhile to note is that the previous formula represents a general relation describing either closed form results [90, 122], based on ML [15, 54], or the average execution times evaluated via simulation: in this dissertation we adopted both these last approaches.

In general, any performance model specified as (4.2) needs to be based on a preliminary profiling of the relevant DIA in order to be representative of its performance. Particularly, such profiling must take into account several metrics, like the number of stages and tasks within each stage, their execution times, and dependencies. The natural formalization for stage dependencies is provided by DAGs.

Since the execution of jobs on a sub-optimal VM type may lead to performance disruptions, it is critical to avoid assigning tasks belonging to class i to the wrong VM type $j \neq \tau_i$. Indeed, YARN allows for specifying node labels and partitioning nodes in the cluster according to these labels, then it is possible to enforce this separation. Our configuration statically splits different VM types with this mechanism and adopts within each partition either a further static separation in classes or a work conserving scheduling mode, where idle resources can be assigned to jobs requiring the same VM type. The choice about the scheduling policy within partitions is not critical, since it does not impact on the optimization technique or performance prediction approach. When resources are tightly separated, we can expect the performance estimate to accurately mirror the real system behavior, whilst in work conserving mode the observed performance may improve due to a better overall utilization of the deployed cluster, hence the prediction is better interpreted as a conservative upper bound. Equations (4.2) can be used to formulate the deadline constraints as:

$$T_i \leq D_i, \quad \forall i \in \mathcal{A}. \tag{4.3}$$

In light of the above, we can say that the ultimate goal of the proposed approach is to identify the optimal VM type selection, τ_i, and type of lease and number of VMs, $\nu_i = r_i + d_i + s_i$, for each class i such that the total operating cost is minimized while the deadlines and concurrency levels are met.

The reader is referred to Figure 4.3 for a graphical overview of the main elements of the considered resource provisioning problem. Furthermore, Table 4.1 reports a complete list of the parameters used in the models presented in the next sections, whilst Table 4.2 summarizes the decision variables.

4.3 Design Time Problem

In the following, we present the optimization models and techniques exploited by the D-SPACE4Cloud tool in order to rightsize the cluster deployment given the profiles characterizing the DIAs under study, the candidate VM types, possibly at different cloud providers, and the prices practiced by those providers. The resulting optimization model is a resource allocation problem, presented in Section 4.3.1.

Table 4.1
Model parameters

Param.	Definition
$\mathcal{A}$	Set of application classes
$\mathcal{V}$	Set of VM types
h_i	Number of concurrent users for class i
Z_i	Class i think time [s]
D_i	Deadline associated to applications of class i [s]
η_i	Maximum percentage of spot VMs allowed to class i
Γ_j	Number of CPUs available within a VM of type j
ρ_j	Effective hourly price for reserved VMs of type j [€/h]
δ_j	Unit hourly cost for on demand VMs of type j [€/h]
σ_j	Unit hourly cost for spot VMs of type j [€/h]
R_{ij}	Number of reserved VMs of type j devoted to class i users

Table 4.2
Decision variables

Var.	Definition
ν_i	Number of VMs assigned for the execution of applications from class i
r_i	Number of reserved VMs booked for the execution of applications from class i
d_i	Number of on demand VMs assigned for the execution of applications from class i
s_i	Number of spot VMs assigned for the execution of applications from class i
c_i	Total number of cores assigned to class i
x_{ij}	Binary variable equal to 1 if VM type j is assigned to application class i

The first issue D-SPACE4Cloud has to solve is to quickly, yet with an acceptable degree of accuracy, estimate the completion times and operational costs of the cluster: to this end, within the mathematical programming formulation of the problem, we decided to exploit ML models for the assessment of application execution times. In this way, it is possible to swiftly explore several plausible configurations and point out the most cost-effective among the feasible ones. Afterwards, the required resource configuration can be fine tuned using more accurate, yet more time consuming and computationally demanding, simulations to obtain a precise prediction of the expected running time.

According to the previous considerations, the first step in the optimization procedure consists in determining the most cost-effective resource type for each application based on their price and the expected performance. The mathematical programming models that allow to identify such an initial solution are discussed in Sections 4.3.1 and 4.3.2. Finally, the algorithm adopted to efficiently tackle the resource provisioning problem is described in Sec-

tion 4.3.3.

4.3.1 Mathematical Programming Model

The big data cluster resource provisioning problem can be formalized as the following mathematical programming formulation:

$$\min_{\mathbf{x},\mathbf{c},\mathbf{r},\mathbf{d},\mathbf{s}} \quad \sum_{i\in\mathcal{A}} \left(\rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i\right) \tag{P1a}$$

subject to:

$$\sum_{j\in\mathcal{V}} x_{ij} = 1, \quad \forall i \in \mathcal{A}, \tag{P1b}$$

$$R_{i,\tau_i} = \sum_{j\in\mathcal{V}} R_{ij} x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1c}$$

$$\rho_{\tau_i} = \sum_{j\in\mathcal{V}} \rho_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1d}$$

$$\delta_{\tau_i} = \sum_{j\in\mathcal{V}} \delta_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1e}$$

$$\sigma_{\tau_i} = \sum_{j\in\mathcal{V}} \sigma_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1f}$$

$$\Gamma_{\tau_i} = \sum_{j\in\mathcal{V}} \Gamma_j x_{ij}, \quad \forall i \in \mathcal{A}, \tag{P1g}$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in \mathcal{A}, \forall j \in \mathcal{V}. \tag{P1h}$$

$$(\mathbf{c},\mathbf{r},\mathbf{d},\mathbf{s}) \in \arg\min \sum_{i\in\mathcal{A}} \left(\rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i\right) \tag{P1i}$$

subject to:

$$r_i \le R_{i,\tau_i}, \quad \forall i \in \mathcal{A}, \tag{P1j}$$

$$s_i \le \frac{\eta_i}{1-\eta_i}(r_i + d_i), \quad \forall i \in \mathcal{A}, \tag{P1k}$$

$$c_i \le \Gamma_{\tau_i}(r_i + d_i + s_i), \quad \forall i \in \mathcal{A}, \tag{P1l}$$

$$\mathcal{T}_i(c_i, h_i; Z_i, \tau_i) \le D_i, \quad \forall i \in \mathcal{A}, \tag{P1m}$$

$$c_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1n}$$

$$r_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1o}$$

$$d_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}, \tag{P1p}$$

$$s_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}. \tag{P1q}$$

Problem (P1) is presented as a bi-level resource allocation problem where the outer objective function (P1a) considers running costs. For each application class the logical variable x_{ij} is set to 1 if the VM type j is assigned to

application class i. We will enforce that only $x_{i,\tau_i} = 1$ in (P1b), thus determining the optimal VM type τ_i for application class i. Hence the following constraints, ranging from (P1c) to (P1g), pick the values for the inner problem parameters.

The inner objective function (P1i) has the same expression as (P1a), but in this case the prices ρ_{τ_i}, δ_{τ_i}, and σ_{τ_i} are fixed, as they have been chosen at the upper level, thus making the function linear in the number of allocated VMs. Constraints (P1j) bound the number of reserved VMs that can be concurrently started according to the contracts in place with the IaaS provider. The subsequent constraints, (P1k), enforce that spot instances do not exceed a fraction η_i of the total assigned VMs and constraints (P1l) link the total available cores with the allocated VMs. Further, constraints (P1m) mandate to respect the deadlines D_i. In the end, all the remaining decision variables are taken from the natural numbers set, according to their interpretation.

4.3.2 Initial Solution

The presented formulation of Problem (P1) is particularly difficult to tackle, as it is a MINLP problem, possibly nonconvex, depending on $\mathcal{T}_i$. According to the literature about complexity theory [50], integer programming problems belong to the NP-hard class, hence the same applies to (P1). However, since there is no constraint linking variables belonging to different application classes, the general formulation can be split into several smaller and independent problems, one per class $i \in \mathcal{A}$:

$$\min_{c_i, r_i, d_i, s_i} \quad \rho_{\tau_i} r_i + \delta_{\tau_i} d_i + \sigma_{\tau_i} s_i \tag{P2a}$$

subject to:

$$r_i \leq R_{i,\tau_i}, \tag{P2b}$$

$$s_i \leq \frac{\eta_i}{1-\eta_i}(r_i + d_i), \tag{P2c}$$

$$c_i \leq \Gamma_{\tau_i}(r_i + d_i + s_i), \tag{P2d}$$

$$\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} \leq D_i, \tag{P2e}$$

$$c_i \in \mathbb{N}, \tag{P2f}$$

$$r_i \in \mathbb{N}, \tag{P2g}$$

$$d_i \in \mathbb{N}, \tag{P2h}$$

$$s_i \in \mathbb{N}. \tag{P2i}$$

In Problem (P2) we rewrote (P1m) as constraint (P2e), which is a simple model of the average execution time, function of the concurrency level and the available containers, among other features, used to enforce that the completion time meets the arranged deadline. On the other hand, the constraints (P2b)–(P2d) were just carried over from the full formulation.

Equation (P2e) is the result of a ML process to get a first order approximation of the execution time of Hadoop and Spark jobs in cloud clusters.

Following Ataie et al. [15], which compares linear regression, Gaussian SVR, polynomial SVR with degree ranging between 2 and 6, and linear SVR, here we opt for a model derived with linear SVR. This is due to the fact that SVR with other kinds of kernel fares worse than the linear one, whilst plain linear regression requires an ad hoc data cleaning to avoid linear dependencies in the design matrix, thus making it harder to apply in the greatest generality. In order to select a relevant feature set, we started by generalizing the analytical bounds for MapReduce clusters proposed in Malekimajd et al. [90] and Verma et al. [122]. This approach yielded a diverse collection of features including the number of tasks in each map or reduce phase, or stage in the case of Spark applications, average and maximum values of task execution times, average and maximum shuffling times, dataset size, as well as the number of available cores, of which we consider the reciprocal. More details about the ML method and the choice of features can be found in Section 3.6.2. Since most of these features characterize the application class, but cannot be controlled, equation (P2e) collapses all but h_i and c_i, with the corresponding coefficients, into a single constant term, χ^0_{i,τ_i}, that is the linear combination of the feature values with the SVR-derived weights.

Problem (P2) can be reasonably relaxed to a continuous formulation as in other literature approaches: for example see Ardagna et al. [10]. Furthermore, the problem can be additionally simplified with a simple algebraic transformation: given the total number of VMs needed to support the required workload, it is trivial to determine the optimal instance mix using dominance considerations. Indeed, since $\sigma_{\tau_i} < \rho_{\tau_i} < \delta_{\tau_i}$, spot VMs are selected first, but respecting the constraint (P2c), then it is the turn of reserved ones, whose number is bounded by R_{i,τ_i} and, at last, on demand VMs will cover the still unheeded computational needs. Moving from this consideration, it is possible to reduce the problem to a formulation that involves only the number of cores, c_i, and the overall number of VMs, ν_i, as exposed below:

$$\min_{c_i,\nu_i} \nu_i \tag{P3a}$$

subject to:

$$c_i \leq \Gamma_{\tau_i} \nu_i, \tag{P3b}$$

$$\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} \leq D_i, \tag{P3c}$$

$$c_i \geq 0, \tag{P3d}$$

$$\nu_i \geq 0. \tag{P3e}$$

The continuous relaxation makes it possible to apply the KKT conditions, which are necessary and sufficient for optimality due to the regularity of Problem (P3), thanks to Slater's constraint qualification: (P3c) is the only nonlinear constraint and is convex in the domain, which in turn contains an internal point. Notice that, in this way, we can analytically obtain the optimum of all the instances of Problem (P3), one per class $i \in \mathcal{A}$, as proven in Theorem 4.3.1.

Theorem 4.3.1. *The optimal solution of Problem* (P3) *is:*

$$c_i = \frac{\chi^c_{i,\tau_i}}{D_i - \chi^h_{i,\tau_i} h_i - \chi^0_{i,\tau_i}}, \tag{4.4a}$$

$$\nu_i = \frac{c_i}{\Gamma_{\tau_i}} = \frac{1}{\Gamma_{\tau_i}} \frac{\chi^c_{i,\tau_i}}{D_i - \chi^h_{i,\tau_i} h_i - \chi^0_{i,\tau_i}}. \tag{4.4b}$$

Proof. The Lagrangian of Problem (P3) is given by:

$$\begin{aligned} \mathcal{L}(c_i, \nu_i) = \nu_i &+ \lambda^\Gamma \left(c_i - \Gamma_{\tau_i} \nu_i \right) + \\ &+ \lambda^\chi \left(\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i \right) + \\ &- \lambda^c c_i - \lambda^\nu \nu_i \end{aligned} \tag{4.5}$$

and stationarity conditions lead to:

$$\frac{\partial \mathcal{L}}{\partial \nu_i} = 1 - \Gamma_{\tau_i} \lambda^\Gamma - \lambda^\nu = 0, \tag{4.6a}$$

$$\frac{\partial \mathcal{L}}{\partial c_i} = \lambda^\Gamma - \lambda^\chi \chi^c_{i,\tau_i} \frac{1}{c_i^2} - \lambda^c = 0, \tag{4.6b}$$

while complementary slackness conditions are:

$$\lambda^\Gamma \left(c_i - \Gamma_{\tau_i} \nu_i \right) = 0, \quad \lambda^\Gamma \geq 0, \tag{4.7a}$$

$$\lambda^\chi \left(\chi^h_{i,\tau_i} h_i + \chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i \right) = 0, \quad \lambda^\chi \geq 0, \tag{4.7b}$$

$$\lambda^c c_i = 0, \quad \lambda^c \geq 0, \tag{4.7c}$$

$$\lambda^\nu \nu_i = 0, \quad \lambda^\nu \geq 0. \tag{4.7d}$$

Constraint (P3c) requires $c_i > 0$ and, thanks to (P3b), it also holds $\nu_i > 0$, thus $\lambda_c = \lambda_\nu = 0$. Now, equations (4.6a) and (4.6b) can be applied to obtain $\lambda_\Gamma > 0$ and $\lambda_\chi > 0$. So constraints (P3b) and (P3c) are active in every optimal solution, whence we get (4.4a) and (4.4b). □

Since Theorem 4.3.1 provides optima in closed form for Problem (P3), it is straightforward to repeat its algebraic solution for all the pairs class-VM of Problem (P1). The choice of the preferred VM type whereon to run each class is made via the comparison of all the relevant optimal values, selecting by inspection the minimum cost association of classes and VM types.

4.3.3 Optimization Algorithm

The aim of this section is to provide a brief description of the optimization heuristic embedded in D-SPACE4Cloud. It efficiently explores the space of possible configurations, starting from the initial ones obtained via Theorem 4.3.1.

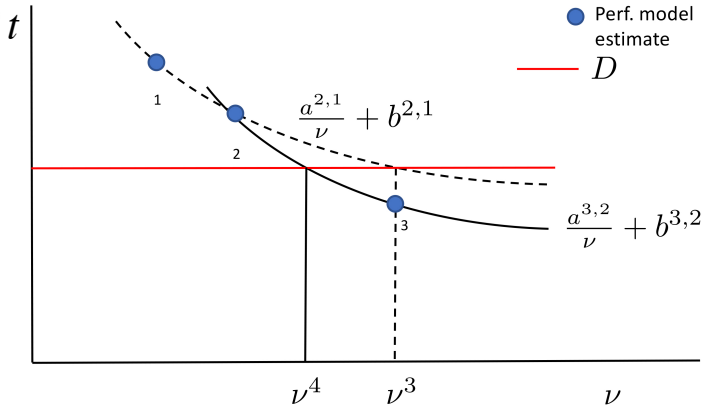


Figure 4.4 – Hyperbolic jump

Since (P3c) might be only a preliminary approximation, the very first step of the procedure is simulating the initial configuration in order to refine the prediction. This step, as well as all the subsequent ones, is executed in parallel, since the original Problem (P1) has been split into independent subproblems. After checking the feasibility of the initial solution, the search algorithm begins the exploration incrementing the VM count whenever the solution results infeasible or decreasing it to save on costs if the current configuration is already feasible.

In order to avoid one-VM steps, which might lead to a very slow convergence for our optimization procedure, particularly when dealing with large clusters, the optimization heuristic exploits the fact that the execution time of DIAs is inversely proportional to the allocated resources [15, 90, 122]. Hence, at every iteration the application execution time is estimated as:

$$t_i = \frac{a_i}{\nu_i} + b_i, \tag{4.8}$$

where t_i is the execution time and ν_i the number of VMs, whilst a_i and b_i are obtained by fitting the hyperbola to the previous steps results. Hence, from the second search step on, we can compute a_i and b_i using the predicted execution times returned by the performance simulators and the associated resource allocations. In this way, at every iteration k it is possible to have an educated guess on the number of VMs required to meet the deadline D_i, as depicted in Figure 4.4, where hyperbolas obtained at subsequent steps are used to determine the next resource allocation to assess by intersection with the line at height D_i. The result of this operation is:

$$\nu_i^{k+1} = \frac{a_i^{k,k-1}}{D_i - b_i^{k,k-1}}. \tag{4.9}$$

Our optimization algorithm aims at combining the convergence guarantees of dichotomic search with the fast exploration allowed by specific knowledge on system performance, such as equations (4.8) and (4.9). Each job class

Algorithm 4.3.1 Local search

Require: $\nu_i^0 \in \mathbb{N}$
1: simulate ν_i^0
2: **if** ν_i^0 is infeasible **then**
3: $\quad \nu_i^1 \leftarrow \nu_i^0 + 1$
4: $\quad l_i^1 \leftarrow \nu_i^0$
5: **else**
6: $\quad \nu_i^1 \leftarrow \nu_i^0 - 1$
7: $\quad u_i^1 \leftarrow \nu_i^0$
8: **end if**
9: **repeat** $k \leftarrow 1, 2, \ldots$
10: $\quad$ simulate ν_i^k
11: $\quad$ update bounds
12: $\quad \nu_i^{k+1} \leftarrow f\left(\nu_i^k, \nu_i^{k-1}\right)$
13: $\quad$ check ν_i^{k+1} against bounds
14: **until** $u_i^k - l_i^k = 1$
15: **return** u_i^k

is optimized separately and in parallel as described in pseudo-code in Algorithm 4.3.1. First off, the initial solution ν_i^0, obtained as outlined in Section 4.3.2, is evaluated using the simulation model. Since equation (4.9) requires at least two points, the conditional at lines 2–8 provides a second point at one-VM distance and sets the initial one as lower or upper bound, according to its feasibility. Then the algorithm searches iteratively the state space performing simulations and keeping track of the interval enclosing the optimal solution. Every new step relies on the hyperbolic function in (4.9), as shown at line 12.

As already mentioned, D-SPACE4Cloud mixes dichotomic search and domain knowledge about performance characteristics in order to exploit the best of both worlds. Fitting a hyperbola to previous results allows to speed up the exploration by directing it where the system performance is expected to be reasonably close to the deadline imposed as constraint, yet the use of only the latest two simulations, dictated by convenience considerations, might hamper convergence with oscillations due to inaccuracies. We address this issue by recording the most resource hungry infeasible solution as lower bound, l_i^k, and the feasible configuration with fewest VMs as upper bound, u_i^k. Hence, at line 11, if ν_i^k turns out to be feasible, then it is assigned to u_i^k, otherwise to l_i^k. Furthermore, every new tentative configuration ν_i^{k+1} predicted at line 12 must belong to the open interval $\left(l_i^k, u_i^k\right)$ to be relevant: at line 13 the algorithm enforces this behavior, falling back to the mid point when this property does not hold.

Now, given the monotonic dependency of execution times on the number of assigned computational nodes, the stopping criterion at line 14 guarantees that the returned configuration is the provably optimal solution of the inner, separate Problem (P2) for class i. In other words, the joint selection of the VM type and their number is NP-hard, but when the type of VM is fixed in the

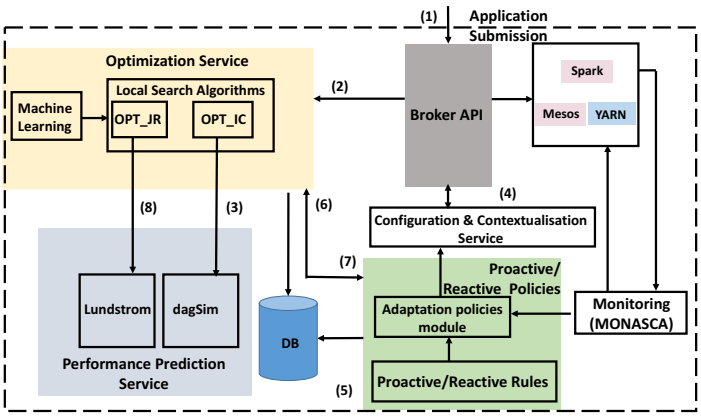


Figure 4.5 – Overall EUBra-BIGSEA architecture

first phase, our heuristic obtains the final solution for all classes in polynomial time.

4.4 BIGSEA Run Time Architecture

In the collaboration with the EUBra-BIGSEA project we improved upon the results of the previous work at design time and contributed to the development of a middleware focused on the run time management of DIAs. This was made possible as EUBra-BIGSEA targets container-based systems, thus enabling a higher degree of flexibility in resource reconfiguration and rebalancing. Furthermore, this project takes into account Apache Mesos[4] as scheduler, again improving in flexibility in comparison to YARN and adding the possibility of negotiation among resource managers.

The run time architecture presented in this section consists of a set of key components designed to satisfy a group of functional requirements: specifically i) handling the resource provisioning, ii) reducing costs related to big data application execution, and iii) guaranteeing QoS. All the modules are driven by the optimization of the resources infrastructure usage. This requirement is accomplished by minimizing costs, monitoring, and dynamically allocating resources to meet deadlines.

The considered Spark applications can be executed on Mesos or, alternatively, on YARN resource managers. Given an application, we characterize its deadline as *hard* or *soft*: hard deadlines must be met, whilst soft deadlines have an associated priority and can be violated if the system does not have enough capacity. In this case, the system tries to minimize the weighted tardiness of soft deadline applications, i.e., it will possibly reallocate the cluster nodes in a way that the weighted sum of application excess time with respect to soft deadlines is minimized.

As shown in Figure 4.5, the architecture consists of a broker and configuration service, a monitoring system, a performance prediction service, and two software management layers, i.e., the optimization service and the proac-

[4] `http://mesos.apache.org`

tive policies module. While the monitoring system fulfills the task of collecting information regarding the system metrics, such as batch queues capacity, network and CPU load, or applications elapsed time, the other layers have distinct goals. The broker, the contextualization service, and the proactive policies module implement the specification and enact high level proactive rules triggering infrastructure adaptation and load balancing. On the other hand, the optimization service aims at pursuing the respect of QoS guarantees and reduction of resource usage costs.

More precisely:

Broker: The Broker API's task consists in receiving application requests enriched with additional information concerning, e.g., the configuration and deadlines, as well as in triggering the application execution.

Optimization Service: The Optimization Service is made up of two sub-components: i) a module able to provide the initial capacity configuration (OPT_IC) and ii) a module to rebalance application allotments in case of heavy load (OPT_JR).

OPT_IC exploits a lookup table including a set of initial solutions, i.e., the optimal configuration in terms of number of cores and VMs to meet a deadline. In case the application runs on a new scenario, the available history is used to perform a linear interpolation, providing as result a guess at the minimum cost configuration. Following this phase, an optimization algorithm to compute the optimal allocation of cores is executed off line and the lookup table is updated correspondingly. You may notice that this is basically a subproblem of (P1) where the VM is fixed.

Performance Prediction Service: The Performance Prediction Service is composed of two predictor tools and estimates application execution times given the total number of available cores.

Specifically, the algorithm used by the initial optimizer exploits an ad hoc DES called *dagSim*, which provides off line accurate results at the cost of long execution times, in the order of seconds or minutes when dealing with complex DAGs. On the other hand, the algorithm used by the re-balancer tool deploys a different predictor called *Lundstrom*, which provides on line less accurate results though performing well in terms of execution time (sub-second) even for large DAGs [8].[5]

Reactive and Proactive policies: This module provides a bridge between users' application submission and the execution platforms, through adding or removing resources according to thresholds triggered by the monitoring infrastructure. Violation of the policy thresholds triggers the execution of high level rules resulting in the adaptation of the infrastructure by horizontally or vertically scaling the currently deployed cores and balancing the load among the physical servers. This module is in charge of controlling the infrastructure and exploits the prediction service in order to understand if an application is early or late with respect to the deadline, since some rules are meant to avoid tardy applications and add resources when the delay goes over a threshold.

[5]The description of both tools can be found at `http://www.eubra-bigsea.eu/menu-deliverables`.

The performance of data analytics applications running on the EUBra-BIGSEA platform is profiled in advance, so a QoS guarantee is defined and resource allocation is optimized according to the performance requirements. Application profiling collects relevant information about stages and their tasks, as said in Section 4.2.

Since optimization-based policies are effective only when application performance profiles are available, the developed techniques consider mainly batch applications, such as data acquisition, bus trajectory identification, and social data clustering, while interactive applications are managed through reactive and proactive rule-based policies. Rules of this kind, specified by system administrators, are based on the analysis of the metrics provided by the EUBra-BIGSEA monitoring system and will trigger reconfigurations if the monitoring metric, for reactive rules, or a prediction of its future value, in the case of proactive rules, is above or below a given threshold.

The work flow for running applications on the above described infrastructure is detailed in the following, with an explanation of the several steps numbered in Figure 4.5. Each submission (1) has to provide a series of relevant data: an application identifier, the dataset size, the required deadline with its type, either soft or hard, plus some further parameters. Based on these pieces of information, the optimization module looks up the application signature in the system's history. Either the optimal configuration is already cached, or the module computes a preliminary guess via interpolation of the available historic data. In the latter case, OPT_IC (3) is used to refine such guess and return an optimal initial configuration, relying on the performance prediction service.

At this point, the broker checks the current system status: under light load conditions the application can enter the system right away, but in heavy load the overall capacity is not enough to fit all the required resources, hence soft deadline applications are rebalanced to accommodate the new submission. The optimal reconfiguration is obtained via OPT_JR (8).

Given the final allocation, the configuration and contextualization service takes care of properly setting up the resource pools (4). Now, the policies module takes over the control of the cluster, with reactive rules that intervene (5) depending on the metrics evaluated by the monitoring service. Similarly, proactive policies poll the optimization service (6) to assess the current configuration and, based on the prediction, identify predicted violations or over-provisioning. Under any of these circumstances, the optimization service updates the requirements (7), thus activating the adaptation module to properly redistribute resources.

4.5 Run Time Problem

In line with the overall architecture requirements, the optimization module has to implement two main functionalities. First, given i) the application signature, ii) the target VM type, established beforehand by system administrators, and iii) the deadline for the big data application execution, required by the end user, the optimization module computes the minimum cost configuration in terms of number of VMs to allocate. In addition, the optimization

service should also rearrange resource pools when the system sustains heavy load and capacity becomes a tight constraint.

At submission time, if the capacity available in the system is enough to accommodate the deployment of a new application, then execution can start. When this is not the case, the IaaS infrastructure will try to achieve the best possible QoS by prioritizing the allocation of resources to hard deadline applications and reallocating the residual capacity in a way that the weighted sum of soft deadline applications' exceeding times is minimized. In such a situation, the system is under heavy load and the optimization module has to determine the new resource configuration, based on these premises. In particular, given i) the soft deadline applications signatures and ii) their current execution progress, the optimization service determines a new configuration that minimizes the weighted tardiness. We are working in the assumption that each soft deadline application is associated with a weight that formalizes its (or its end user's) priority in the eye of the EUBra-BIGSEA platform: the higher the weight, the more important is to meet the prearranged SLAs.

Note that, in this second scenario, the newly submitted application can start its execution only borrowing resources from soft deadline applications, hence it will receive the amount of resources established in the first scenario if and only if i) it is a hard deadline application and ii) the shared cluster has enough capacity to accommodate all the hard deadline applications' requirements. Specifically, if the latter condition cannot be satisfied, the submission is rejected and system administrators are alerted of the failure to accept the job.

Optimizing the initial configuration is substantially just another application of the single class sizing problem, Problem (P2), discussed in Sections 4.3.2 and 4.3.3. Section 4.5.1 introduces the approach for the weighted tardiness scenario. Later, Section 4.5.2 analyzes the formulation and derives the closed form optimal solution of its continuous relaxation. In the end, Section 4.5.3 introduces a search procedure aimed at determining an optimal integral solution for the minimum weighted tardiness problem.

4.5.1 Mathematical Programming Model

If the system is under heavy load, the optimization module has to reallocate cluster resources. First of all, hard deadline applications cannot be delayed, so their share of VMs is assigned with priority. As soon as these requirements are fulfilled, the residual capacity can be allocated among soft deadline applications so as to minimize their overall weighted tardiness. Please note that in this scenario, since we are operating at run time, it is not possible to change VM types: rather, each application i is associated with specific instances τ_i, according to the results of the design time problem.

Let us denote with $\mathcal{A}^{\mathrm{d}}$ the set of soft deadline applications. As in the previous section, each application is characterized by the parameters obtained via ML techniques, χ^{c}_{i,τ_i} and χ^{0}_{i,τ_i}, the deadline D_i, and the number of cores available on every VM, Γ_{τ_i}. Since this problem is solved while jobs actually run on the cluster, it is possible to take into account monitoring data to assess whether applications are either executing faster than expected or falling behind schedule. Almeida et al. [7] provides more details, but here it is material to know that the only impact is a rescaling of deadlines, hence we continue

with the already introduced notation. The available spare capacity in terms of cores is N. Each application is also characterized by a weight w_i: the larger the value, the higher is the application priority. Given the hybrid ML formulation for response time, it is possible to define the tardiness of every application as:

$$\chi^{c}_{i,\tau_i}\frac{1}{c_i}+\chi^{0}_{i,\tau_i}-D_i, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{4.10}$$

In this scenario all the application classes have only one concurrent job at a time: equation (4.10) is just constraint (P2e) where $h_i = 1$ and the corresponding term χ^{h}_{i,τ_i} is aggregated into χ^{0}_{i,τ_i}. Since the system is under heavy load, the number of cores allocated to application i is such that the deadline D_i cannot be met and the tardiness is strictly positive.

Again we adopt the decision variables c_i, representing the number of cores, and ν_i, counting the assigned VMs, hence we can derive the following formulation:

$$\min_{\mathbf{c},\boldsymbol{\nu}} \quad \sum_{i\in\mathcal{A}^{\mathrm{d}}} w_i\left(\chi^{c}_{i,\tau_i}\frac{1}{c_i}+\chi^{0}_{i,\tau_i}-D_i\right) \tag{P4a}$$

subject to:

$$c_i \le \Gamma_{\tau_i}\nu_i, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P4b}$$

$$\sum_{i\in\mathcal{A}^{\mathrm{d}}} \Gamma_{\tau_i}\nu_i \le N, \tag{P4c}$$

$$c_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P4d}$$

$$\nu_i \in \mathbb{N}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{P4e}$$

Problem (P4) is a straightforward weighted tardiness minimization under a capacity constraint. The objective function (P4a) is the sum of all the tardiness terms associated to the various applications, each multiplied by the corresponding weight. The set of constraints (P4b) imposes a bound on the number of cores available per VM, while constraint (P4c) enforces that no more than the remaining N cores are allocated to soft deadline applications. In the end, constraints (P4d)-(P4e) state that all the variables c_i and ν_i are taken from the natural numbers, as expected given their interpretation.

4.5.2 Initial Solution

Taking the continuous relaxation of Problem (P4), it is possible to consider KKT conditions both necessary and sufficient for optimality, given the fact that Slater's constraint qualification holds. The relaxed problem is formulated as follows:

$$\min_{\mathbf{c},\boldsymbol{\nu}} \quad \sum_{i\in\mathcal{A}^{\mathrm{d}}} w_i\left(\chi^{c}_{i,\tau_i}\frac{1}{c_i}+\chi^{0}_{i,\tau_i}-D_i\right) \tag{P5a}$$

subject to:

$$c_i \le \Gamma_{\tau_i} \nu_i, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P5b}$$

$$\sum_{i \in \mathcal{A}^{\mathrm{d}}} \Gamma_{\tau_i} \nu_i \le N, \tag{P5c}$$

$$c_i \ge 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{P5d}$$

$$\nu_i \ge 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{P5e}$$

The following Theorem 4.5.1 provides a closed form optimal solution for the continuous problem.

Theorem 4.5.1. *The optimal solution of Problem* (P5) *is:*

$$\begin{cases} c_1 = \dfrac{N}{1+\sum_{j \in \mathcal{A}^d \setminus \{1\}} \sqrt{\frac{w_j}{w_1} \frac{\chi^c_{j,\tau_j}}{\chi^c_{1,\tau_1}}}}, \\ c_i = \dfrac{\sqrt{\frac{w_i}{w_1} \frac{\chi^c_{i,\tau_i}}{\chi^c_{1,\tau_1}}}}{1+\sum_{j \in \mathcal{A}^d \setminus \{1\}} \sqrt{\frac{w_j}{w_1} \frac{\chi^c_{j,\tau_j}}{\chi^c_{1,\tau_1}}}} N, \qquad \forall i \in \mathcal{A}^d \setminus \{1\}, \end{cases} \tag{4.11}$$

with:

$$\nu_i = \frac{c_i}{\Gamma_{\tau_i}}, \quad \forall i \in \mathcal{A}^d. \tag{4.12}$$

Proof. The Lagrangian of Problem (P5) is given by:

$$\begin{aligned} \mathcal{L}(\mathbf{c}, \boldsymbol{\nu}) = &\sum_{i \in \mathcal{A}^{\mathrm{d}}} w_i \left(\chi^c_{i,\tau_i} \frac{1}{c_i} + \chi^0_{i,\tau_i} - D_i \right) + \sum_{i \in \mathcal{A}^{\mathrm{d}}} \lambda^\Gamma_i \left(c_i - \Gamma_{\tau_i} \nu_i \right) + \\ &+ \lambda^N \left(\sum_{i \in \mathcal{A}^{\mathrm{d}}} \Gamma_{\tau_i} \nu_i - N \right) - \sum_{i \in \mathcal{A}^{\mathrm{d}}} \lambda^c_i c_i - \sum_{i \in \mathcal{A}^{\mathrm{d}}} \lambda^\nu_i \nu_i \end{aligned} \tag{4.13}$$

and stationarity conditions lead to:

$$\frac{\partial \mathcal{L}}{\partial c_i} = -\frac{w_i \chi^c_{i,\tau_i}}{c_i^2} + \lambda^\Gamma_i - \lambda^c_i = 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.14a}$$

$$\frac{\partial \mathcal{L}}{\partial \nu_i} = -\lambda^\Gamma_i \Gamma_{\tau_i} + \lambda^N \Gamma_{\tau_i} - \lambda^\nu_i = 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.14b}$$

while complementary slackness conditions are:

$$\lambda^\Gamma_i \left(c_i - \Gamma_{\tau_i} \nu_i \right) = 0, \quad \lambda^\Gamma_i \ge 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.15a}$$

$$\lambda^N \left(\sum_{i \in \mathcal{A}^{\mathrm{d}}} \Gamma_{\tau_i} \nu_i - N \right) = 0, \quad \lambda^N \ge 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.15b}$$

$$\lambda_i^c c_i = 0, \quad \lambda_i^c \geq 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.15c}$$

$$\lambda_i^\nu \nu_i = 0, \quad \lambda_i^\nu \geq 0, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{4.15d}$$

We know that $c_i > 0$ by the definition of tardiness and, according to constraints (P5b), also $\nu_i > 0$ holds. Thanks to the conditions (4.15c) and (4.15d), then, we derive $\lambda_i^c = \lambda_i^\nu = 0$, for all $i \in \mathcal{A}^{\mathrm{d}}$.

From the KKT (4.14a) and (4.14b) we have:

$$\lambda_i^\Gamma = \frac{w_i \chi_{i,\tau_i}^c}{c_i^2}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.16a}$$

$$\lambda_i^\Gamma = \lambda^N, \quad \forall i \in \mathcal{A}^{\mathrm{d}}, \tag{4.16b}$$

whence derive $\lambda_i^\Gamma > 0$, $\forall i \in \mathcal{A}^{\mathrm{d}}$ and $\lambda^N > 0$, thus the constraints (P5b) and (P5c) are active in optimal solutions. Substituting:

$$\lambda^N = \lambda_i^\Gamma = \frac{w_i \chi_{i,\tau_i}^c}{c_i^2}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{4.17}$$

Exploiting (4.17), we obtain also:

$$\frac{w_i \chi_{i,\tau_i}^c}{c_i^2} = \frac{w_j \chi_{j,\tau_j}^c}{c_j^2}, \quad \forall (i,j) \in \mathcal{A}^{\mathrm{d}} \times \mathcal{A}^{\mathrm{d}} \tag{4.18}$$

and:

$$c_i = c_1 \sqrt{\frac{w_i}{w_1} \frac{\chi_{i,\tau_i}^c}{\chi_{1,\tau_1}^c}}, \quad \forall i \in \mathcal{A}^{\mathrm{d}}. \tag{4.19}$$

Since all the (P5b) hold as equalities, we can substitute $\Gamma_{\tau_i} \nu_i$ for c_i in constraint (P5c), which is also active. Using (4.19) we can compute a closed form for c_1 and, consequently, for all the decision variables of this problem, thus obtaining (4.11). □

4.5.3 Optimization Algorithm

Theorem 4.5.1 only provides a solution for the continuous relaxation of Problem (P5). Additionally the ML model used to formulate tardiness and, consequently, the objective function (P4a) is just a first approximation for the system performance. These reasons motivate a search procedure to determine an integer solution and, possibly, leverage the higher accuracy of simulation-based models to attain more efficiency or a decrease in global weighted tardiness. Algorithm 4.5.1 reports this state space exploration: it receives in input the set of soft deadline applications and the maximum number of iterations allowed to run, then it returns the optimized cores allocation. The detailed description of its behavior follows.

First of all, the loop at lines 1–3 obtains the initial continuous solution via the formulas and, right after, the VMs allocation $\boldsymbol{\nu}$ is heuristically made feasible at line 4. The idea is simple: it is enough to round each ν_i to the next

Algorithm 4.5.1 Resource rebalancing

Require: $\mathcal{A}^{\mathrm{d}}$, $\bar{k} \in \mathbb{N}$
1: **for** $i \in \mathcal{A}^{\mathrm{d}}$ **do**
2: compute ν_i as per (4.11)
3: **end for**
4: round $\boldsymbol{\nu}$ and make the configuration feasible
5: **repeat** $k \leftarrow 1, 2, \ldots$
6: $\boldsymbol{\nu}^{\mathrm{old}} \leftarrow \boldsymbol{\nu}$
7: $U \leftarrow \emptyset$
8: **for** $i \in \mathcal{A}^{\mathrm{d}}$ **do**
9: **for** $j \in \mathcal{A}^{\mathrm{d}} : j \neq i$ **do**
10: choose $\Delta\nu_i$ and $\Delta\nu_j$ so that $\Gamma_{\tau_i}\Delta\nu_i = \Gamma_{\tau_j}\Delta\nu_j$
11: evaluate δ_{ij}^{H} with hyperbola, as per (4.8)
12: **if** $\delta_{ij}^{\mathrm{H}} < 0 \wedge \nu_j - \Delta\nu_j \geq 1$ **then**
13: add $\left(\Delta\nu_i, \Delta\nu_j\right)$ to U
14: **end if**
15: **end for**
16: **end for**
17: $\delta_{ij}^{*} \leftarrow 0$
18: $m^{*} \leftarrow (0, 0)$
19: **for** $\left(\Delta\nu_i, \Delta\nu_j\right) \in U$ **do**
20: evaluate δ_{ij}^{L} with Lundstrom
21: **if** $\delta_{ij}^{\mathrm{L}} < \delta_{ij}^{*}$ **then**
22: $m^{*} \leftarrow \left(\Delta\nu_i, \Delta\nu_j\right)$
23: **end if**
24: **end for**
25: apply m^{*} to $\boldsymbol{\nu}$
26: **until** $k \geq \bar{k} \vee \boldsymbol{\nu} = \boldsymbol{\nu}^{\mathrm{old}}$
27: **return** $\boldsymbol{\nu}$

greater integer and, if need be, to remove one VM from the applications with the smallest weights until the capacity constraint (P4c) is satisfied.

At line 5 starts the main loop of the local search algorithm. Each iteration amounts to the exploration of a neighborhood centered around the current optimal solution, with a best improvement policy. The neighborhood contains all the configurations that can be reached if an application gives up one VM and its cores are reassigned to another job, see line 10. Different applications are possibly hosted on different VMs, then it is necessary to make sure that these moves respect the proportionality of vCPUs. For example, if job $\bar{\imath}$ yields 1 quad-core VM and $\bar{\jmath}$ is hosted on single-core instances, the latter will receive 4 replicas, thus leaving unchanged the total resource pool. Since the size of the neighborhood is $O(|\mathcal{A}^{\mathrm{d}}|^2)$, relying solely on simulations for the exploration would make optimization times too long for use at run time. This is the reason why the neighborhood is first swept via the algebraic approximation (4.8) in the two nested loops at lines 8–16. Instead of reevaluating the full objective function for each move, the algorithm only considers the partial

effect on the weighted tardiness values for i and j: these terms are called δ_{ij}. According to the condition at line 12, only the promising moves that do not take all the VMs from an application ($v_j - \Delta v_j \geq 1$) and improve the overall weighted tardiness in the hyperbolic approximation ($\delta_{ij}^{\mathrm{H}} < 0$) are later assessed via simulations.

The second part of the main iteration (lines 17–25) goes over all the promising moves identified with the algebraic approximation and reevaluates them via the more accurate Lundstrom's model. The variable m^*, initially set to the identity move, keeps track of the best improving move. At last, the main cycle concludes by applying the best move and increasing the iteration count. Line 26 states the stopping criterion: either the algorithm keeps exploring for the maximum number of iterations $\bar{k}$, or the latest neighborhood did not provide any improving moves, meaning that the search procedure reached a local minimum.

4.6 Discussion

The material of this chapter consists primarily in two formulations of optimization problems relevant for the design and management of DIAs. In particular, at design time this contribution enables capacity allocation, as well as the choice of optimal resource types, while at run time it focuses on the minimization of tardiness when reacting to workload spikes.

In both cases we proceed in three major steps: first of all a rigorous formulation of the problem at hand, then the exploitation of the KKT conditions to obtain an initial solution in closed form, in the end we devise a search procedure to refine the optimal solution. The performance models presented in Chapter 3 fit into these steps coherently with their peculiarities, with ML that is valuable in the analytical phase thanks to the algebraic formulas it yields, whilst simulation-based models have an edge in the final exploration due to their insensitivity to the parameter range where they are evaluated.

The proposed formulations and optimization procedures show how it is possible to tackle research question 2. Specifically, the design time problem solved by D-SPACE4Cloud allows for determining beforehand the minimum cost deployment for target DIAs, based on the expected incoming load. In addition to this result, the run time problem studied within EUBra-BIGSEA enables reallocating the available resources according to the actual current load, even if this is higher than what previously estimated. This is possible thanks to looser SLAs that permit some submissions to become tardy, rather than imposing a penalty immediately when the completion is not timely.

Alongside its theoretical core, the current chapter also presents the architecture of the systems where the proposed optimization procedures are implemented. On one side, D-SPACE4Cloud is part of the DICE IDE, whose main objective is supporting the adoption of big data technologies with a DevOps approach. On the other hand, in the collaboration with EUBra-BIGSEA we developed OPT_IC and OPT_JR, which focus on the run time management of DIA deployments. As Chapter 6 will detail, the use of D-SPACE4Cloud can grant savings throughout a cluster life cycle of more than 30 % and enable investigating in early design stages the impact of some architectural choices, such as privacy preserving mechanisms. Similarly, OPT_IC and OPT_JR allow

for reacting within minutes to sudden unforeseen changes in the incoming workload.

Relevant Publications

The publications relevant to the optimization techniques discussed in this chapter follow. The design time architecture of D-SPACE4Cloud was described in a tool paper submitted to QUDOS [32], a workshop held during the International Conference on Performance Engineering. More details about both the architecture, Section 4.1, and the design time optimization problem, Section 4.3, are presented in Ciavotta et al. [33]. Another publication, Malekimajd et al. [89], discusses a similar formulation focused on legacy MapReduce jobs. On the other hand, the aspects investigated during the EUBra-BIGSEA project, Sections 4.4 and 4.5, were recently proposed in Ardagna et al. [8].

CHAPTER 5

Performance Models Validation

This chapter presents and discusses several validation experiments carried out to assess the accuracy and effectiveness of the performance models described in Chapter 3. First of all, in Section 5.1 we detail the experimental platforms used for these runs. Later, Section 5.2 introduces validation results for MapReduce, with a particular focus on QNs, SWNs, and fluid techniques. Further on, Section 5.3 overviews the results we obtained to validate dagSim, whilst Section 5.4 lists experiments and results for the hybrid method validation. In the end, in Section 5.5 we show the outcomes obtained in validating the models proposed for CNNs and Section 5.6 wraps up the chapter with a discussion of the results and their relationship with research questions 1 and 4.

5.1 Experimental Platforms

All the data about real systems performance for the validation of the proposed models has been measured on a series of clusters, either rented on public clouds or accessed in private data centers. Namely, we have real logs obtained on Amazon EC2, Microsoft Azure, CINECA—the Italian supercomputing center, as well as some GPU-enabled machines at IBM Research Yorktown. The motivation behind this choice is to verify the robustness of the models against the different contention patterns shown by each platform, as caused by both their setups and other users sharing the same underlying infrastructure.

The Amazon cluster included 30 m4.xlarge instances with a total of 120 vCPUs configured to support 240 containers overall. On the other hand, on Microsoft Azure we provisioned clusters of variable size, reaching up to 26 nodes. In this case we also used different instances, with A4 and D12v2 VMs.

PICO,[1] the big data cluster available at CINECA, is composed of 74 nodes, each of them boasting two Intel Xeon 2670v2 2.5 GHz 10-core processors,

[1] http://www.hpc.cineca.it/hardware/pico

with 128 GB RAM per node. Out of this 74 nodes, up to 66 are available for computation. In our experiments on PICO, we used several configurations ranging from 60 to 120 cores and set up the scheduler to provide one container per core.

The cluster is shared among different users, hence resources are managed by the Portable Batch System (PBS). PBS allows for submitting jobs and checking their progress, configuring at a fine-grained level the computational requirements: for all submissions it is possible to request a number of nodes and to define how many CPUs and what amount of memory are needed on each of them. Since the cluster is shared, the performance of single jobs depends on the overall system load, even though PBS tries to split the resources. Due to this, it is possible to have large variations in performance according to the total usage of the cluster. In particular, storage is not handled directly by PBS, thus leading to an even greater impact on performance.

We tried to mitigate this variability first of all by requesting entire nodes of the cluster for the execution of our experiments. In such a way, we could be sure that nobody else could run other jobs on the same nodes, thus interfering with the performance measurement. An ephemeral Hadoop cluster has been created at the beginning of each experiment on the allocated nodes, where we also locally kept HDFS, in order to experience lower variability than the one observed using the centralized storage. In spite of these settings, still the experiments showed high variability, in particular with a few runs characterized by extremely high execution time. In our analysis we discarded runs with an anomalous execution time, taking out all the experiments that lie more than three standard deviations away from the average computed for the same configuration.

The dataset used for testing has been generated using the TPC-DS benchmark[2] data generator, creating at a scale factor ranging from 250 GB to 1,000 GB several files directly used as external tables by Hive or Spark. We chose the TPC-DS benchmark as it is the industry standard for benchmarking data warehouses.

5.2 Performance Models for MapReduce

As MapReduce benchmark, we introduced five ad hoc queries,[3] which are mapped as a single MapReduce job in Hive. This queries are named R1–5 and are shown in Listing 5.1. The profiling phase has been conducted by extracting average task durations from at least twenty runs of each query. The discussed parameters are shown in Table 5.1, where we show the values obtained from measurements performed on CINECA. In the table we report the scale factor f, the number of tasks of each phase, respectively n^{M} and n^{R} for mappers and reducers, and the average task durations, respectively $\bar{t}^{\mathrm{M}}$ and $\bar{t}^{\mathrm{R}}$. As you can see, the number of tasks in each phase can grow in the order of thousands.

[2] http://www.tpc.org/tpcds/

[3] https://github.com/deib-polimi/Hive-Experiment-Runner

```
select avg(ws_quantity), avg(ws_ext_sales_price),
       avg(ws_ext_wholesale_cost), sum(ws_ext_wholesale_cost)
from web_sales
where (web_sales.ws_sales_price between 100.00 and 150.00)
   or (web_sales.ws_net_profit between 100 and 200)
group by ws_web_page_sk
limit 100
```

(a) R1

```
select inv_item_sk, inv_warehouse_sk
from inventory
where inv_quantity_on_hand > 10
group by inv_item_sk, inv_warehouse_sk
having sum(inv_quantity_on_hand) > 20
limit 100
```

(b) R2

```
select avg(ss_quantity), avg(ss_net_profit)
from store_sales
where ss_quantity > 10 and ss_net_profit > 0
group by ss_store_sk
having avg(ss_quantity) > 20
limit 100
```

(c) R3

```
select cs_item_sk, avg(cs_quantity) as aq
from catalog_sales
where cs_quantity > 2
group by cs_item_sk
```

(d) R4

```
select inv_warehouse_sk, sum(inv_quantity_on_hand)
from inventory
group by inv_warehouse_sk
having sum(inv_quantity_on_hand) > 5
limit 100
```

(e) R5

Listing 5.1 – Interactive queries

5.2.1 QN and SWN Models

The QN and SWN models presented in Chapter 3 have been validated with an experimental campaign on Amazon EC2 and CINECA, the Italian supercomputing center. Section 5.1 details the two setups. The target framework was Hadoop 2.6.0.

We used the GreatSPN [16] and JMT [24] tools for the performance analysis of SWNs and QNs, respectively. In both cases the simulation stopping criterion was configured with 10 % accuracy at a 95 % confidence interval. Via parsing Hadoop logs it is possible to obtain lists of task execution times, which are needed for the replayer in JMT service centers. These logs are also used to choose a proper distribution and parameters for the *map* and *reduce* transitions in the SWN models. As discussed in Section 3.1, while reduce task service times can be considered exponentially distributed, the execution time of map tasks fits better with more general distributions, like Erlang [9]. In

Table 5.1
Fitted parameters, CINECA

Query	f [GB]	n^{M}	$\bar{t}^{\mathrm{M}}$ [ms]	n^{R}	$\bar{t}^{\mathrm{R}}$ [ms]
R1	250	144	25,970	151	2,346
R1	500	287	32,159	300	1,958
R1	750	434	34,244	455	1,996
R1	1,000	591	40,534	619	3,063
R2	250	4	57,326	4	8,785
R2	500	2	42,202	2	6,582
R2	750	3	48,869	3	7,500
R2	1,000	65	1,082,274	68	13,086
R3	250	381	28,659	400	2,369
R3	500	757	37,018	793	2,570
R3	750	1,148	42,348	1,009	2,785
R3	1,000	1,560	41,961	1,009	3,048
R4	250	288	25,087	302	2,958
R4	500	573	41,007	601	2,961
R4	750	868	43,902	910	3,259
R4	1,000	1,183	42,615	1,009	8,667
R5	250	4	13,456	4	1,424
R5	500	2	11,774	2	1,499
R5	750	3	12,682	3	1,462
R5	1,000	64	19,557	68	1,610

particular, we used Erlang-2 for R1, Erlang-4 for R2 and R3, lastly Erlang-5 for R4 and R5. The shape and rate parameters are set according to each query profile. The other timed transitions appearing in the SWN models are considered to be exponentially distributed. Note that the choice of a specific PDF for the execution times of different tasks depends on the query, but also on the underlying system: contention and other performance disrupting behaviors influence the timings empirically measured on the various platforms.

The following results validate the QNs and SWNs discussed in Sections 3.2 and 3.3. We feed the models with parameters evaluated via the experimental setup and compare the measured response times with the simulated ones. Specifically, we consider as a quality index the accuracy on response time prediction. We define errors as:

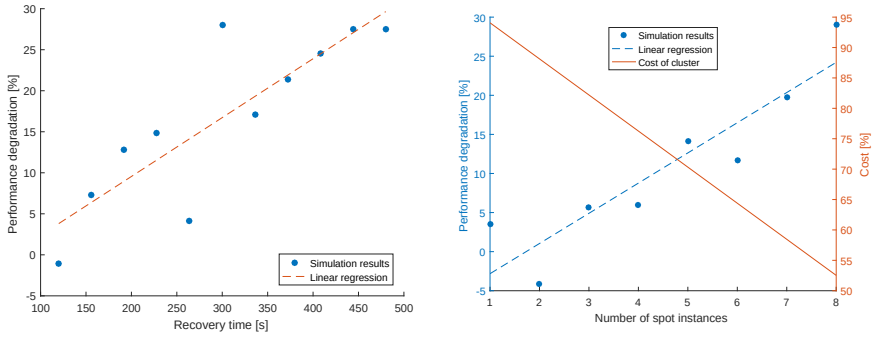
$$\varepsilon = \frac{T^{\mathrm{s}} - T^{\mathrm{r}}}{T^{\mathrm{r}}}, \tag{5.1}$$

where T^{s} is the simulated response time, whilst T^{r} is the average measurement across multiple runs. Such a definition allows not only to quantify the relative error on execution times, but also to identify cases where the predicted time is smaller than the actual one, thus leading to possible deadline misses. Indeed, if $\varepsilon < 0$ then the prediction is not conservative.

Table 5.2 lists the validation results for QNs and SWNs. Among these experiments, we considered both single user scenarios, repeatedly running the same query on a dedicated cluster with $Z = 10\,\mathrm{s}$, and multiple user scenarios, with the same think time, but higher concurrency levels. For all the experiments we report the number of concurrent users, the overall cores available in the cluster, the dataset scale factor, and the total number of map and reduce

TABLE 5.2
QN AND SWN MODELS ACCURACY

Query	Provider	h	c	f [GB]	n^{M}	n^{R}	T^{r} [ms]	$\varepsilon_{\mathrm{QN}}$ [%]	$\varepsilon_{\mathrm{SWN}}$ [%]
R1	Amazon	1	240	250	500	1	55,410	−8.40	−8.63
R2	Amazon	1	240	250	65	5	36,881	−25.45	2.97
R3	Amazon	1	240	250	750	1	76,806	0.60	8.48
R4	Amazon	1	240	250	524	384	92,197	−14.72	−3.01
R1	CINECA	1	60	500	287	300	378,127	8.94	−12.69
R3	CINECA	1	100	500	757	793	401,827	30.59	26.36
R3	CINECA	1	120	750	1,148	1,009	661,214	14.82	5.61
R4	CINECA	1	60	750	868	910	808,490	4.48	−0.26
R3	CINECA	1	80	1,000	1,560	1,009	1,019,973	−1.00	0.03
R5	CINECA	1	80	1,000	64	68	39,206	−6.65	−1.04
R1	CINECA	3	20	250	144	151	1,002,160	3.67	−9.27
R1	CINECA	5	20	250	144	151	1,736,949	−30.02	−17.74
R2	CINECA	3	20	250	4	4	95,403	17.45	4.00
R2	CINECA	5	20	250	4	4	145,646	−32.97	3.09
R1	CINECA	5	40	250	144	151	636,694	3.70	−3.63
R2	CINECA	3	40	250	4	4	86,023	22.97	−17.81
R2	CINECA	5	40	250	4	4	90,674	13.78	29.68



(a) Against recovery time

(b) Varying the number of spot instances

Figure 5.1 – Performance degradation and cost reduction

tasks, plus the mean execution time measured on the real system and the relative errors for both models. In the worst case, the relative error can reach up to 32.97 %, which is in line with the expected accuracy in the performance prediction field [78]. Moreover, the SWN model achieves a higher accuracy, with the average of the relative errors taken in absolute value that decreases from the 14.13 % of QNs down to 9.08 %.

5.2.2 Spot Failure Analysis

To evaluate the SWN model of Figure 3.3 we considered query R1 running on the 1,000 GB dataset with 15 VMs, 4 cores each. Without failures, the execution time of R1 is 556,680 ms, where around 57 % of the time is spent in the map stage. The baseline simulation time is T_0 = 533,438 ms, yielding a −4.18 % relative error. We define the performance degradation as $\eta(t)$ =

$(T(t)-T_0)/T_0$, where $T(t)$ is the simulated response time obtained via the SWN model in Figure 3.3 when the recovery time is t. To measure the recovery time of our system on Amazon EC2, we switched off one of the VMs, started a new one after 30 s and recorded the moment when the first task is assigned for execution to the new instance. In ten experiments, the mean recovery time was equal to 331,715 ms, where the time to start the new instance was around 180 s.

In the first analysis we consider a conservative scenario where the failure is injected into the system in an early stage of query execution, so we fixed the mean time to failure to 50 s. We considered a cluster with 7 spot VMs out of 15 and we varied the recovery time between 120 s and 480 s. In this way we estimate system performance degradation in a range where VMs startup is either faster (for example, in container-based systems where the startup time is negligible and the recovery time is due only to YARN NodeManagers startup) or slower than on Amazon EC2. The plot reported in Figure 5.1a shows, as expected, that the longer the recovery process stalls, the more performance degrades, and it can reach up to 25 % if the recovery process takes 8 minutes.

Note that in some cases we obtained negative values for $\eta(t)$ because simulation data is subject to inaccuracies. This is why we rely on linear regression to estimate the overall performance degradation trend. In this and the next analysis we obtained a p-value equal to 0.001 for the full model F-test, meaning that the regression line is a good fit except for white noise.

In the second experiment, the mean recovery time is fixed to 331,715 ms, the average we measured in our experiments, and the number of spot instances is increased from 1 to 7 out of the total of 15 VMs. The result is shown in Figure 5.1b, which reports the results for cost reduction due to using spot instances as well as performance degradation. Cost reduction is computed as $C_r = C(s;p)/C(0;0)$, where $C(s;p)$ is the cluster cost when s spot instances are used, with a failure probability p. $C(0;0)$ is the case when no spot instance is used. $C(s;p)$ can be computed as:

$$C(s;p) = \delta(\nu - s) + p\delta s + (1-p)\sigma s, \tag{5.2}$$

where ν is the total number of VMs, while δ and σ are hourly prices for on demand and spot instances, respectively. δ and σ are set to \$0.285 and \$0.031,3, based on Amazon's pricing.

It should be noted that, according to the experiments, the job execution time is shorter than an hour: according to Amazon's pricing policy, in case spot instances are abruptly terminated, then users are not charged for partial hour usage. The first term on the right hand side of (5.2) is the cost of the initial on demand instances, while the second term is the price the user should pay to acquire the same number of on demand instances as the lost spot ones in case of failure. Finally, with probability $(1-p)$ spot instances do not fail and the third term indicates the cost in this scenario. We evaluate the probability of spot termination as $p = \mathbb{P}(X < T)$, where X is the random variable for time to failure and T is the execution time. Assuming X is exponentially distributed, p can be easily obtained in closed form. Figure 5.1b shows that, based on the proposed SWN model, one can efficiently use spot instances to decrease cluster costs, with a 52.5 % saving for 8 spot instances in exchange for just a 25 % performance degradation.

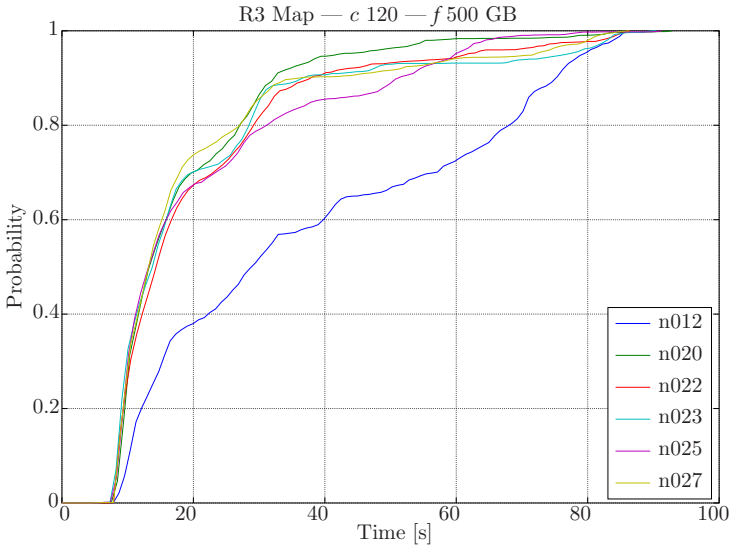


Figure 5.2 – R3 map, 120 containers, 500 GB dataset

5.2.3 Fluid Models

In this section, we describe the results of the experiments we conducted to validate the fluid techniques described in Section 3.4. These experiments have been performed at CINECA, with the configuration described in Section 5.1.

First of all, we studied the empirical cumulative distribution functions (CDFs) of task durations on different nodes, in order to identify the cases in which performance was strongly affected by exogenous interference. We then considered the accuracy that can be reached adopting the approach based on a convex combination of the deterministic and exponential limiting cases.

Figures 5.2, 5.3, 5.4 and 5.5 show the empirical CDFs derived from the measurements. Figures 5.2 and 5.3 refer to subsequent phases of the same experimental run, namely, query R3 running on the six-node, 120-container cluster deployment over the 500 GB dataset. Accordingly, Figures 5.4 and 5.5 show query R5 on the three-node, 60-container cluster over the 750 GB dataset.

As a general trend well represented in Figures 5.3 and 5.5, reducers tend to behave quite regularly. Most likely, possible variabilities spread across the shuffle stage that overlaps with the map phase, hence end up being hardly noticeable in each reducer task duration. On the other hand, mappers suffer a stronger impact from external interference. In the reported graphics we have examples of both good and problematic performance. Figure 5.2 shows how all but one node have a very similar behavior. Further, the only different node is not significantly distant from the others, hence probably the low variability is imputable to the physiological effects of data locality. Instead, Figure 5.4 reports that the three involved nodes are quite variable in performance. In this case, since we are dealing with a small interactive query, any exogenous interference may cause a strong impact on the overall response time, thus making prediction harder.

Tables 5.3 and 5.4 reports the results of the accuracy assessment for the

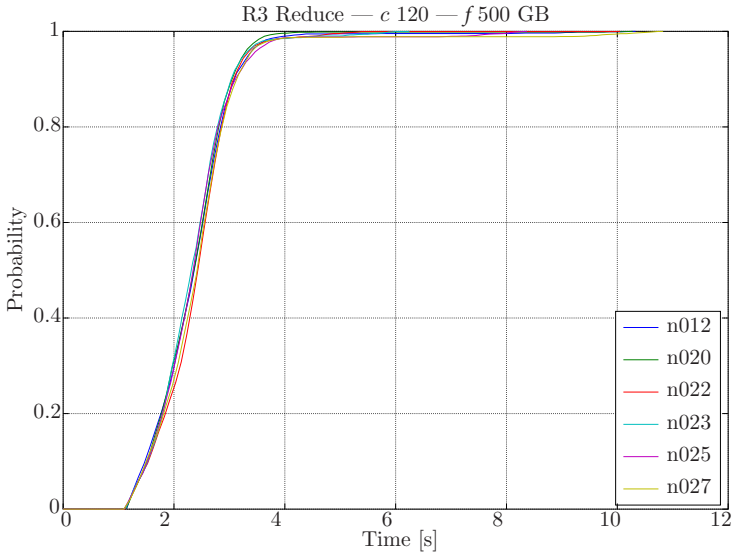


Figure 5.3 – R3 reduce, 120 containers, 500 GB dataset

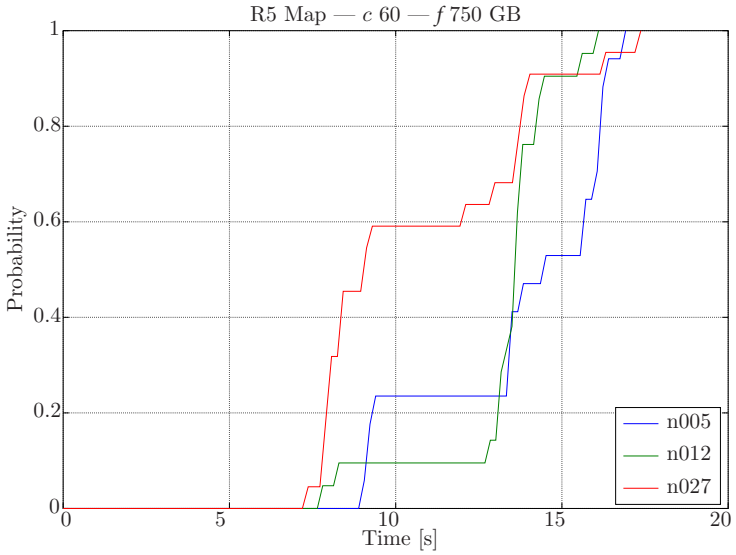


Figure 5.4 – R5 map, 60 containers, 750 GB dataset

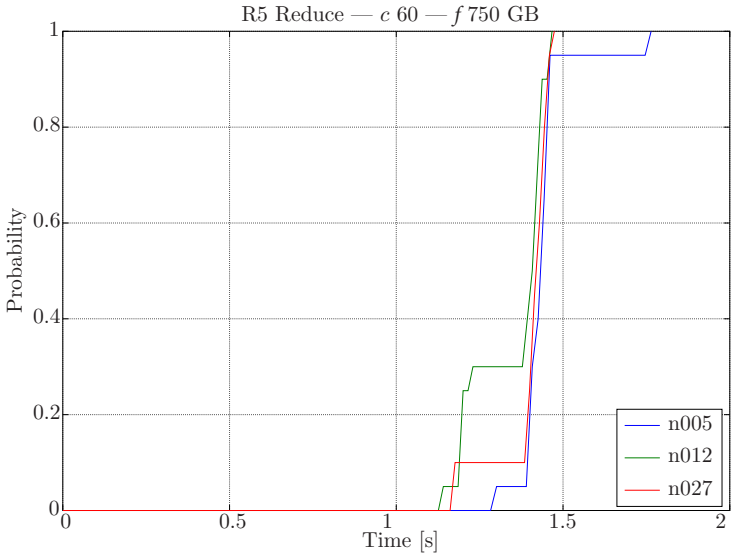


Figure 5.5 – R5 reduce, 60 containers, 750 GB dataset

proposed method, considering MapReduce jobs only. In particular, we consider the fluid evolution function (3.7), discussed in Section 3.4.4. For several experiments we report the involved query, the total number of containers available in the cluster (c), the scale factor for the dataset generator (f), the average measured (T^r) and predicted (T^s) execution time, and the relative error (ε). The average accuracy achieved with the approximate formula is 9.23 %, perfectly in line with the expectations in the performance prediction field [78], where a 30 % accuracy on response times is acceptable. Only a handful of experiments show a relatively high error, peaking at 28.91 % in absolute value. Nonetheless, the standard error of the mean is 1.33 %. The largest relative errors tend to appear in the experiments involving R5, a small query both in terms of number of tasks and overall duration. This behavior suggests that the relative abundance of resources might amplify the effects of variability, making prediction a harder task to accomplish.

5.3 DagSim Models

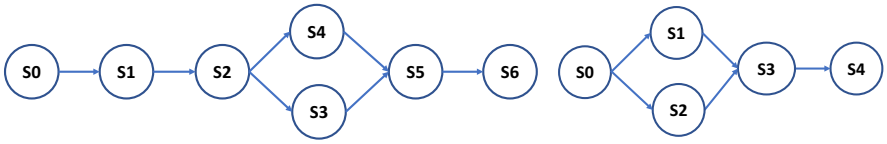
In order to validate dagSim, an ad hoc DES specifically developed for modeling DAG-based DIAs, we collected real measures by running SQL queries on Apache Spark.[4] The techniques adopted by dagSim itself are not presented in Chapter 3 because the tool was devised by project partners, yet it is available as third party simulator in both D-SPACE4Cloud and the EUBra-BIGSEA modules, hence the following results are instrumental to assess the precision of their predictions.

The dataset was generated with TPC-DS, as discussed in Section 5.1. Listing 5.2 shows the considered queries: Q26 and Q52, which belong to the TPC-

[4] `https://spark.apache.org`

Table 5.3
Accuracy with the fluid approximate formula, part 1 of 2

Query	c	f [GB]	T^r [ms]	T^s [ms]	ε [%]
R1	60	250	80,316	82,314	2.49
R2	60	250	84,551	78,100.96	−7.63
R3	60	250	275,684	282,737.30	2.56
R4	60	250	219,243	221,205.13	0.89
R5	60	250	25,924	19,134.99	−26.19
R1	60	750	389,562	387,158.42	−0.62
R2	60	750	80,090	74,975.87	−6.39
R3	60	750	1,027,329	1,052,517.40	2.45
R4	60	750	808,490	834,140	3.17
R5	60	750	24,392	17,339.16	−28.91
R1	60	1,000	556,680	564,379.06	1.38
R2	60	1,000	2,009,929	2,546,188	26.68
R3	60	1,000	1,374,024	1,419,558.01	3.31
R4	60	1,000	1,374,244	1,562,961.30	13.73
R5	60	1,000	48,839	60,180	23.22
R1	80	500	143,139	138,049.55	−3.56
R2	80	500	73,243	69,092.11	−5.67
R3	80	500	526,760	533,398.48	1.26
R4	80	500	410,376	423,152	3.11
R5	80	500	23,558	17,090.94	−27.45
R1	80	750	268,821	266,099.64	−1.01
R2	80	750	78,080	73,276.07	−6.15
R3	80	750	791,314	807,333.48	2.02
R4	80	750	618,045	634,756.77	2.70
R5	80	750	23,894	17,100.00	−28.43
R1	80	1,000	439,052	442,487.10	0.78
R2	80	1,000	1,110,685	1,129,076.31	1.66
R3	80	1,000	1,019,973	1,043,506.48	2.31
R4	80	1,000	960,985	1,061,851.90	10.50
R5	80	1,000	39,206	28,972.61	−26.10



(a) Q26 **(b)** Q52

Figure 5.6 – Spark queries DAGs

DS benchmark. These queries have been executed on SparkSQL, yielding the DAGs shown in Figure 5.6.

Since profiles collect statistical information about jobs, we repeated the profiling runs at least twenty times per query. Analogously to the MapReduce case, properly parsing the logs allows to extract all the parameters composing every query profile, for example average and maximum task execution times, number of tasks, etc. Profiling has been performed on Microsoft Azure, with A4 and D12v2 VMs, with clusters of variable sizes, reaching up to 26 dual-

Table 5.4
Accuracy with the fluid approximate formula, part 2 of 2

Query	c	f [GB]	T^r [ms]	T^s [ms]	ε [%]
R1	100	250	49,726	59,324	19.30
R2	100	250	82,861	77,078.03	−6.98
R3	100	250	168,209	172,945.69	2.82
R4	100	250	138,238	141,341.11	2.24
R5	100	250	25,316	18,281.41	−27.79
R1	100	500	132,383	127,974.74	−3.33
R2	100	500	73,870	69,572.08	−5.82
R3	100	500	401,827	416,948.55	3.76
R5	100	500	24,619	18,226.33	−25.97
R1	100	750	203,531	203,550.04	0.01
R2	100	750	78,291	73,068.51	−6.67
R3	100	750	635,991	651,107.92	2.38
R4	100	750	514,310	526,617.13	2.39
R5	100	750	24,887	17,988.72	−27.72
R1	120	250	46,215	50,790	9.90
R2	120	250	83,136	76,901.37	−7.50
R3	120	250	143,650	144,140.92	0.34
R4	120	250	97,829	90,498.44	−7.49
R5	120	250	26,072	18,720.46	−28.20
R1	120	500	91,809	83,580.16	−8.96
R2	120	500	72,543	68,232.80	−5.94
R3	120	500	303,843	298,283.35	−1.83
R4	120	500	275,407	279,794.24	1.59
R5	120	500	25,265	18,030.72	−28.63
R1	120	750	199,234	200,260.15	0.52
R2	120	750	79,042	74,269.17	−6.04
R3	120	750	661,214	660,269.38	−0.14
R4	120	750	507,861	513,599.16	1.13
R5	120	750	24,882	18,143.36	−27.08

core containers. Along with profiles, we also collected lists of task execution times to feed the replayer in dagSim stages.

Among these experiments, we considered single user scenarios, where one query has been run repeatedly on a dedicated cluster, interleaving a 10 s average think time between completions and subsequent submissions. Tables 5.5 and 5.6 report the results for dagSim models for single user scenarios on the 500 GB dataset. Alongside the query name, real and predicted execution times and the relative error, these tables also list the overall number of available CPUs and the total number of tasks (n) involved in each query. Due to the different shapes of the DAGs, n is the number of tasks across all the stages of each query. The worst case error is −19.01 % and, on average, errors settle at 3.06 %.

5.4 Hybrid Models

In this section we explore in depth the behavior of our hybrid ML modeling technique when it is applied to MapReduce and Spark applications, while also

```
select i_item_id,
       avg(cs_quantity) agg1,
       avg(cs_list_price) agg2,
       avg(cs_coupon_amt) agg3,
       avg(cs_sales_price) agg4
from catalog_sales, customer_demographics, date_dim,
     item, promotion
where catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
  and catalog_sales.cs_item_sk = item.i_item_sk
  and catalog_sales.cs_bill_cdemo_sk = customer_demographics.cd_demo_sk
  and catalog_sales.cs_promo_sk = promotion.p_promo_sk
  and cd_gender = 'F'
  and cd_marital_status = 'W'
  and cd_education_status = 'Primary'
  and (p_channel_email = 'N' or p_channel_event = 'N')
  and d_year = 1998
group by i_item_id
order by i_item_id
limit 100
```

(a) Q26

```
select dt.d_year, item.i_brand_id brand_id,
       item.i_brand brand, sum(ss_ext_sales_price) ext_price
from date_dim dt, store_sales, item
where dt.d_date_sk = store_sales.ss_sold_date_sk
  and store_sales.ss_item_sk = item.i_item_sk
  and item.i_manager_id = 1
  and dt.d_moy=12
  and dt.d_year=1998
group by dt.d_year, item.i_brand, item.i_brand_id
order by dt.d_year, ext_price desc, brand_id
limit 100
```

(b) Q52

Listing 5.2 – TPC-DS queries Q26 and Q52

discussing its extrapolation and interpolation capabilities in each case.

The dataset used for running the experiments has been generated using the TPC-DS benchmark data generator, as detailed in Section 5.1. The scale factor was set to 250 GB. For MapReduce experiments we deployed Hadoop 2.5.1, while for Spark applications version 1.6.0 was considered. Moreover, all these experiments were executed on PICO, CINECA's big data cluster, for a total of about 20,000 CPU hours.

In SVR training, weights are used as a means to suggest the ML to give more relevance and trust to real samples, rather than synthetic ones. Therefore, all the experiments to validate the hybrid approach adopt a five to one ratio between real and analytical data: even if some operational data points might be noisy, higher weights assigned to real data allow for achieving better accuracy than pure AMs, since the effect of noisy data is managed within the inner loop. The number of inner iterations of Algorithm 3.6.1 was set to 10. We used GNU Octave for numerical computation, relying on LibSVM [29] as ML library.

To validate the effectiveness of the proposed hybrid approach in a comparative manner, we introduce two techniques that do not exploit AMs in learning:

Basic ML relies on SVR for the computation of the regression function. In

Table 5.5
DagSim model validation, Microsoft Azure D12v2

Query	c	n	T^{r} [ms]	T^{s} [ms]	ε [%]
Q26	12	1,406	660,700	620,773	−6.04
Q52	12	704	658,397	654,464	−0.60
Q26	16	1,406	551,669	495,246	−10.23
Q52	16	704	515,202	512,122	−0.60
Q26	20	1,406	454,054	393,414	−13.36
Q52	20	704	410,588	407,066	−0.86
Q26	24	1,406	385,639	332,364	−13.81
Q52	24	704	356,296	353,852	−0.69
Q26	28	1,406	354,183	286,861	−19.01
Q52	28	704	302,741	299,305	−1.13
Q26	32	1,406	304,048	250,327	−17.67
Q52	32	704	263,034	260,648	−0.91
Q26	36	1,406	244,214	228,456	−6.45
Q52	36	704	245,084	242,489	−1.06
Q26	40	1,406	225,484	208,327	−7.61
Q52	40	704	213,353	211,291	−0.97
Q26	44	1,406	198,966	189,840	−4.59
Q52	44	704	198,044	196,234	−0.91
Q26	48	1,406	186,659	186,953	0.16
Q52	48	704	188,860	187,162	−0.90
Q26	52	1,406	170,516	171,346	0.49
Q52	52	704	177,380	175,511	−1.05

this case, the algorithm is fed with the same operational data used by our hybrid ML at the last iteration.

Iterative ML adds operational data iteratively and the initial KB is empty, lacking the AM information. In other terms, we considered the general structure of Algorithm 3.6.1 except lines 1 and 2 that corresponds to AMs' involvement.

To compare quantitatively our hybrid approach with the basic and iterative ML baseline methods, we define three performance measures:

The MAPE of response time focuses on prediction accuracy. It is defined as the mean of relative errors, taken as percentages, of predicted response times against their expected value measured on the operational system. MAPE is used to evaluate both extrapolation and interpolation capabilities.

The number of iterations of the external loop of Algorithm 3.6.1, which equals to the number of real data samples fed into the ML model.

The cost of the experiments performed in the cloud in order to obtain operational data. In formula:

$$p \sum_{i \in \mathcal{C}} n_i c_i \bar{T}_i, \tag{5.3}$$

Table 5.6
DagSim model validation Microsoft Azure A3

Query	c	n	T^{r} [ms]	T^{s} [ms]	ε [%]
Q26	6	1,406	2,475,150	2,479,524.66	0.18
Q52	6	704	2,101,121	2,094,742.67	−0.30
Q26	8	1,406	2,014,112	2,026,360.58	0.61
Q52	8	704	1,651,055	1,644,624.56	−0.39
Q26	10	1,406	1,718,490	1,720,192.80	0.10
Q52	10	704	1,270,516	1,258,821.03	−0.92
Q26	12	1,406	1,632,222	1,647,299.29	0.92
Q52	12	704	1,067,327	1,059,946.54	−0.69
Q26	14	1,406	1,381,072	1,393,737.17	0.92
Q52	14	704	918,809	913,134.69	−0.62
Q26	16	1,406	1,213,972	1,224,156.94	0.84
Q52	16	704	827,597	823,043.16	−0.55
Q26	18	1,406	1,069,438	1,095,197.38	2.41
Q52	18	704	759,571	752,902.94	−0.88
Q26	20	1,406	1,036,132	1,035,922.42	−0.02
Q52	20	704	681,948	676,836.53	−0.75
Q26	22	1,406	919,943	989,514.49	7.56
Q52	22	704	608,599	603,718.41	−0.80
Q26	24	1,406	850,542	872,744.23	2.61
Q52	24	704	561,149	556,509.52	−0.83
Q26	26	1,406	657,342	671,679.98	2.18
Q52	26	704	507,889	504,324.45	−0.70
Q52	28	704	474,160	470,658.54	−0.74
Q26	30	1,406	586,840	625,812.40	6.64
Q26	32	1,406	565,578	579,209.02	2.41
Q26	34	1,406	561,356	583,397.80	3.93
Q52	34	704	397,761	392,896.90	−1.22
Q26	36	1,406	511,154	536,921.19	5.04
Q52	36	704	377,816	374,978.62	−0.75
Q26	38	1,406	482,202	507,705.68	5.29
Q52	38	704	375,542	373,554.58	−0.53
Q26	40	1,406	466,190	491,614.12	5.45
Q52	40	704	354,247	351,353.37	−0.82
Q26	42	1,406	425,101	447,379.84	5.24
Q52	42	704	329,417	327,510.50	−0.58
Q26	44	1,406	406,187	429,318.34	5.69
Q52	44	704	321,173	316,978.11	−1.31
Q26	46	1,406	383,123	391,511.05	2.19
Q52	46	704	314,163	316,043.45	0.60
Q26	48	1,406	367,084	398,411.46	8.53
Q52	48	704	300,379	296,947.31	−1.14

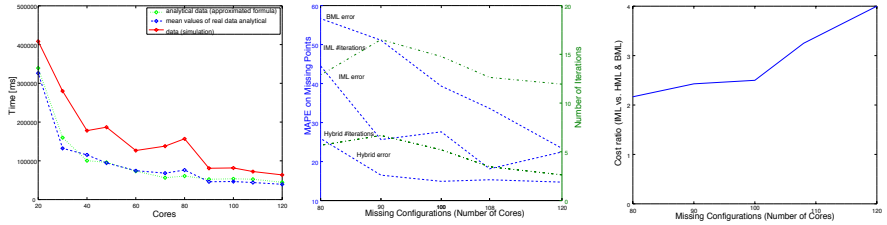


Figure 5.7 – (a) Comparison of formula-based approximation, simulation, and the mean values of real data, (b) right extrapolation, and (c) cost for R1 query

where $\mathcal{C}$ is the set of different configurations considered for model selection and training, n_i the number of repetitions of each experiment, c_i the number of cores associated to each particular configuration, and $\bar{T}_i$ the mean execution time measured for single runs in configuration i. Moreover, p is the time unit cost of CPUs practiced by the cloud vendor. In layman's terms, this is the sum of CPU hours devoted to experiments multiplied by the hourly cost of cloud instances.

The following sections describe the experiments. In order to fairly compare our hybrid approach with the iterative ML, we describe how to find optimal thresholds that minimize errors on response time. Then, the extrapolation and interpolation capabilities of the approaches are investigated when some points lack from the set of available configurations, i.e., they are used only to eventually test model accuracy, without contributing to the training or cross validation. It is important to point out that, while for MapReduce scenarios the thresholds were optimized both for the hybrid and iterative approach, the subsequent experiments to validate accuracy for Spark jobs compare iterative ML run with optimal thresholds to hybrid models that still rely on the ones obtained for MapReduce. In this way the baseline method attains its best performance, whilst the proposed approach is used in a realistic situation, with thresholds optimized on historical data that is not necessarily similar to the new dataset.

5.4.1 MapReduce Job Analysis with Approximate Formula

In this section, we report the results we have achieved on the R1 MapReduce query (see Listing 5.1a) considering the approximate formula (3.15) as AM technique. Several configurations ranging from 20 to 120 cores have been used for this set of experiments. For each configuration, the profiling phase has been conducted extracting the number of map and reduce tasks and their average durations across twenty runs.

The configuration set for analytical data includes 11 different numbers of cores used for executing MapReduce jobs. Figure 5.7a plots the average response time of MapReduce job executions versus the number of cores. The average relative error of the values obtained from equation (3.15) is around 16 % with respect to the mean values of real samples, the maximum relative error is about 31 %.

In order to have a fair comparison between the approaches, we found the optimal combination of the (τ^{in}, τ^{out}) pair in both the hybrid and iterative ML cases. We consider optimal the values that minimize MAPE on the V_2 set.

We varied τ^{in} in the range $[25, 40]$ and τ^{out} in $[10, 30]$ with step 1.[5] For every combination of the two thresholds, the algorithms are run with 50 seeds for the pseudorandom number generator, so as to have different outcomes. Then the generated results are averaged to compute the mean value of MAPE across seeds for every combination. This method showed that the optimum thresholds are $(34, 23)$ and $(30, 19)$ for the hybrid and iterative ML approaches, respectively.

5.4.1.1 Extrapolation Capability on Many Cores

To examine the extrapolation capability of the approaches in the upper region of the configuration set, the following analyses progressively remove from the training set and cross validation sets V_1 and V_2 the configurations with the largest available capacity, which are moved to the test set. In other terms, in the first scenario the training and V_1 set included AM data and the real system data only for configurations from 20 to 100 cores, while the V_2 and test sets included experimental data for 108 and 120 cores, respectively. In the second case, we used in the training and V_1 set real data from 20 to 90 cores, the V_2 set included operational data for 100 cores, while the test set included experimental results in the range $[108, 120]$ cores and so on. Then, the error on response time prediction, the number of iterations, and the associated cost of the alternative techniques were compared.

If not differently stated in the following, both the extrapolation and interpolation capabilities analyses are performed by training 50 models obtained setting 50 different seeds. These are also different from the ones used to identify the optimal thresholds.

As can be seen in Figure 5.7b, the error on the test set of the hybrid and the iterative ML across the 50 models are close up to three missing points, i.e., three configurations excluded during the training phase; but when the number of missing points grows, our hybrid approach performs better than the iterative ML. On the other hand, the number of learning iterations of the hybrid approach is between 2 and 6 and is thus rather smaller than the one of iterative ML, which varies between 12 and 16.5, on average. As Figure 5.7c shows, the iterative ML cost is from 2 to 4 times larger than our hybrid technique, based on the increased number of iterations needed for convergence, hence the larger amount spent for experiment on the real system.

5.4.1.2 Extrapolation Capability on a Few Cores

Next, we examined the extrapolation capability of the approaches in the lower region of the configuration set. In the beginning, the test set includes only 20-core data then, moving from the left side of the configuration axis towards the right, we gradually added other points. As it can be seen from Figure 5.8, our hybrid approach outperforms both the iterative ML and basic ML approaches in terms of the MAPE in almost all scenarios. Specifically, as long as the number of missing points is small, the errors on the test set of the hybrid and

[5] Here we dropped the percent signs to keep the notation light and easier to follow.

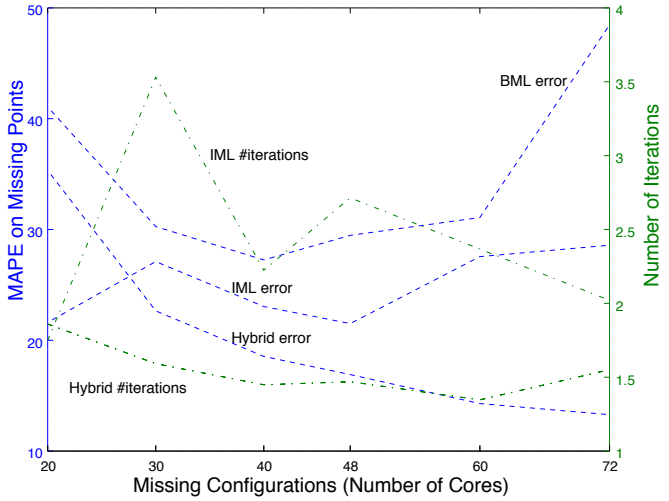


Figure 5.8 – Left extrapolation for R1 query

iterative ML are relatively close, but the number of iterations of the hybrid approach is much smaller than for iterative ML. When the number of missing points gradually grows, although the number of iterations of the hybrid and iterative ML models get closer, the accuracy of the hybrid approach improves in comparison with the iterative ML.

Comparing Figure 5.8 and Figure 5.7b, you can see that all the alternative approaches yield higher errors when extrapolating towards the region with a low resource allocation. We can enumerate a few reasons for this behavior. First, the left side of the response time curve is more informative than the right side, as depicted in Figure 5.7a. As a result, the prediction when some configurations on the left end of the axis are excluded from the training set is more difficult. Second, the optimization process for finding the optimal combinations of the thresholds was run including 120 in the V_2 set, thus naturally favoring better results at the higher end of the axis. All in all, these experiments show an error too large for common performance evaluation practice [78], since they reach higher than 30 %. As the behavior repeats also for the other considered technologies, we will omit the analogous analyses in the next sections. However it is common practice for DIAs to be deployed on reasonably large clusters, hence investigating the extrapolation behavior towards greater numbers of supporting cores is the most sensible choice.

5.4.1.3 Interpolation Capability

To assess the interpolation capability of our proposed approach, we considered three different scenarios: a configuration where the training and V_1 set included analytical and operational data for: (i) 20, 72, and 120 cores, (ii) 20, 48, 72, 100, and 120 cores, and (iii) 20, 40, 48, 72, 80, 100, and 120 cores. These scenarios are reported in Figure 5.9 and Figure 5.10 on the x-axis with values 3, 5, and 7, which represent the number of configurations excluded in

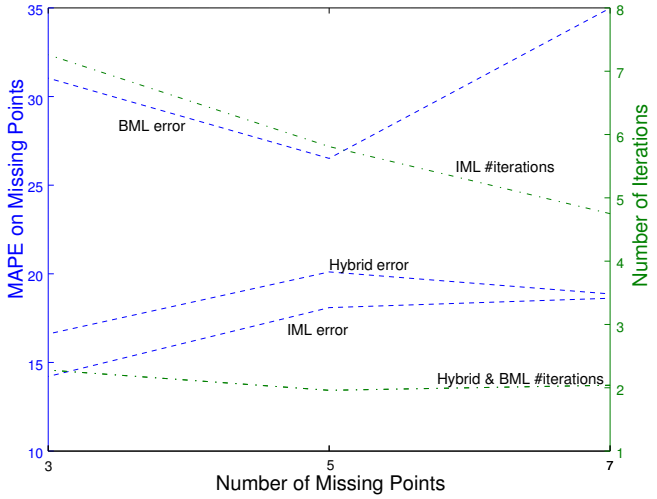


Figure 5.9 – Interpolation for R1 query

the learning phase.

As shown in Figure 5.9, though the error on response time prediction of the hybrid approach is slightly worse than that of the iterative ML, the hybrid approach still performs much better than basic ML. Furthermore, the number of iterations of the hybrid approach is between 2 and 3 in all three scenarios, thus smaller than iterative ML's. Moreover, as shown in Figure 5.10, the cost of constructing hybrid and basic ML models is much less than that of the iterative ML; for example, it is almost one third of the iterative ML's cost when three or five points are missing. Hence our hybrid approach performs better than the basic ML in terms of accuracy and outperforms the iterative ML in terms of the number of experiments to run and the corresponding cost.

5.4.2 MapReduce Job Analysis with QN Simulation

The goal of this section is to evaluate our hybrid approach when relying on a less accurate AM, i.e., QN simulation, with respect to the previous section. The lower accuracy was obtained by considering exponential time distribution for map and reduce stages, while the best fit for the map stage are Erlang PDFs (see Ardagna et al. [9] for further details). In this way, we can verify if our hybrid approach is too sensitive to the accuracy of the AM or if the interpolation and right extrapolation capabilities can be obtained also in such conditions.

At first, the QN model shown in Figure 3.1 is used to generate the set of synthetic data samples that constitute the initial KB. This was done with JMT [24], setting up a 95 % confidence interval at 10 % accuracy. We set the think time, Z, to 10 seconds in a single user scenario.

In Figure 5.7a, the average response times obtained from simulation (red line) are compared with those obtained from real experiments. The average relative error of the values observed from simulations is around 65 % with

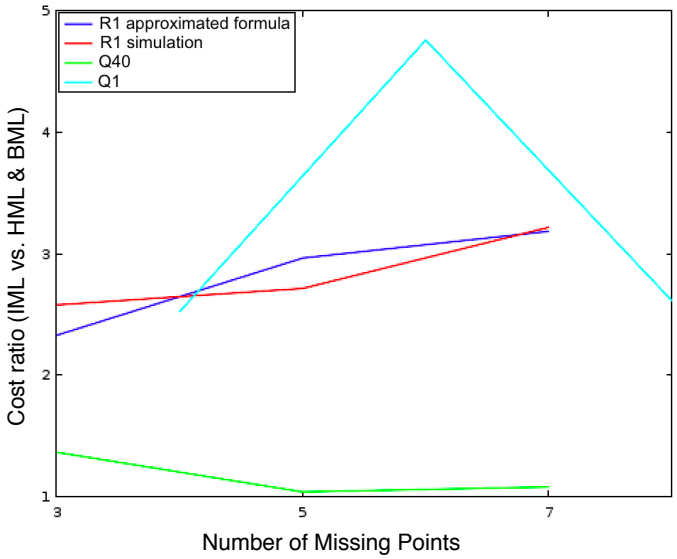


Figure 5.10 – Cost of interpolation analyses results

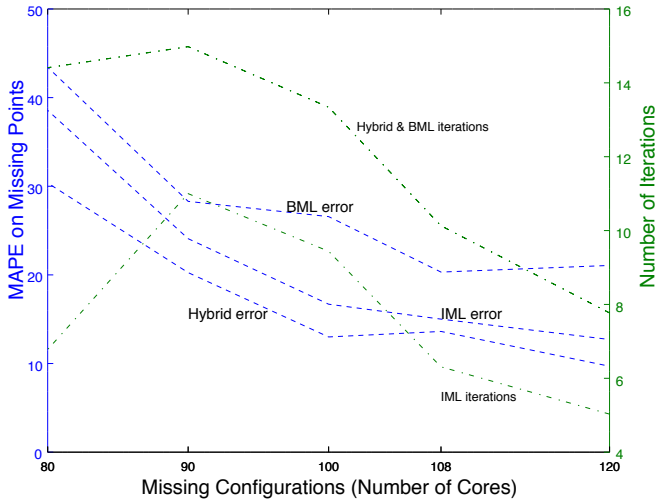


Figure 5.11 – Right extrapolation for R1 query (simulation)

respect to the mean values of real samples, and in the worst case the relative error reaches 96 %, which shows that the QN analysis is very conservative.

To have a fair comparison between the two techniques, the optimal combination of the (τ^{in}, τ^{out}) thresholds was determined as in the previous section both for our hybrid and the iterative ML approach. The optimal thresholds are (38, 24) and (30, 19) for the hybrid and iterative ML approaches, respectively.

The results of the right extrapolation are reported in Figure 5.11. Our

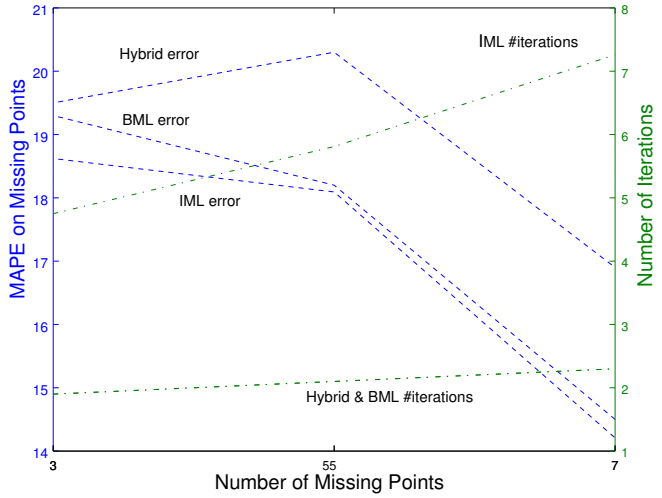


Figure 5.12 – Interpolation for R1 query (simulation)

hybrid approach outperforms the iterative ML approach in terms of the MAPE in all scenarios. When the only missing configuration is 120, the error of our hybrid approach is around 11 %, in contrast to around 14 % of the iterative ML. On the other hand, when half of the rightmost points of the configuration set are missing, the hybrid approach achieves 30 % error on the test set while the iterative ML obtains about 38 %.

For assessing the interpolation capability of our proposed approach, we followed the same procedure described in Section 5.4.1.3. As shown in Figure 5.12, the error on response time prediction of the hybrid approach is slightly worse than the one of iterative ML and basic ML, while its number of iterations is significantly lower than for iterative ML. The maximum relative error in the case of hybrid ML reaches 20.5 %, while for iterative and basic ML are 19.5 % and 18.5 %, respectively. Moreover, as shown in Figure 5.10, the cost of hybrid and basic ML models is much lower, up to three times, than the iterative ML model's.

5.4.3 Spark Job Analysis with Approximate Formula

In order to examine the predictions techniques on Apache Spark, we performed the last set of experiments on the official Q40 query (Listing 5.3) of the TPC-DS benchmark, whose DAG is shown in Figure 5.13, using the approximate formula in equation (3.15) as AM. In particular, we followed a similar process as reported in Section 5.4.1, using the same set of configurations for both analytical and real data as in the R1 case. The profiling phase has been conducted extracting the number of tasks and the average task durations from around ten runs with the same configuration.

In Figure 5.14a we compare the average execution times measured on the real system to the ones obtained with equation (3.15). The average relative error of the approximation formula is around 34 %.

```
select w_state, i_item_id,
   sum(case
       when (cast(d_date as date) < cast('1998-04-08' as date))
       then cs_sales_price - coalesce(cr_refunded_cash, 0)
       else 0
       end) as sales_before,
   sum(case
       when (cast(d_date as date) >= cast('1998-04-08' as date))
       then cs_sales_price - coalesce(cr_refunded_cash, 0)
       else 0
       end) as sales_after
from catalog_sales left outer join catalog_returns
  on (catalog_sales.cs_order_number = catalog_returns.cr_order_number
 and catalog_sales.cs_item_sk = catalog_returns.cr_item_sk),
     warehouse, item, date_dim
where i_current_price between 0.99 and 1.49
  and item.i_item_sk = catalog_sales.cs_item_sk
  and catalog_sales.cs_warehouse_sk = warehouse.w_warehouse_sk
  and catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
  and date_dim.d_date between '1998-03-09' and '1998-05-08'
group by w_state, i_item_id
order by w_state, i_item_id
limit 100
```

Listing 5.3 – TPC-DS query Q40

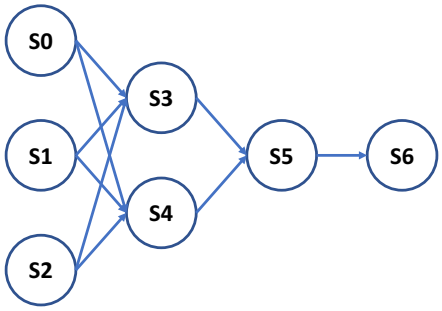


Figure 5.13 – Q40 DAG

The optimal pair (τ^{in}, τ^{out}) was determined only for the iterative ML approach and was (25, 15). Vice versa, in the case of hybrid approach, we used the same thresholds (34, 23) obtained for the R1 query in Section 5.4.1.

Right extrapolation capability analysis results are reported in Figure 5.14 (b) and (c). Figure 5.14b shows that the use of the proposed approach defeats the iterative ML, always providing a lower MAPE on the test set. What is remarkable is that while we move to the left side, where more points are excluded in learning, the iterative ML's MAPE increases dramatically, demonstrating the dominance of our proposed approach. In particular, when only the 120 configuration is missing, the error of the hybrid model is approximately 7 %, contrary to 15 % of the iterative ML. However, when 5 points are missing from the configuration set, the error of iterative ML shoots up to 30 % while in the case of hybrid algorithm just a small increase is observed, reaching 11 %.

Concerning interpolation, we considered three different scenarios when applying the hybrid algorithm to query Q40: (i) three points are missing—20, 72, and 120 cores, (ii) five points—20, 48, 72, 100, and 120 cores, and finally

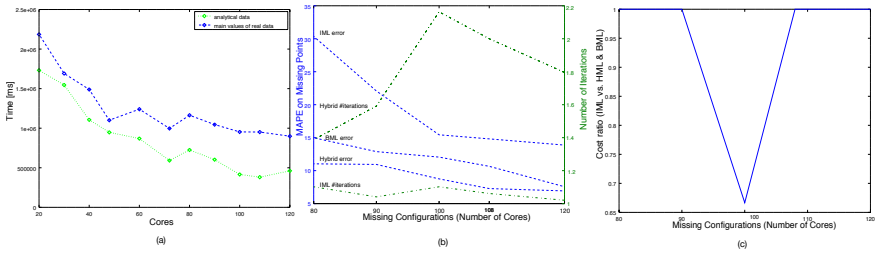


Figure 5.14 – (a) Comparison of formula-based approximation and the mean values of real data, (b) right extrapolation and (c) cost for Q40 query

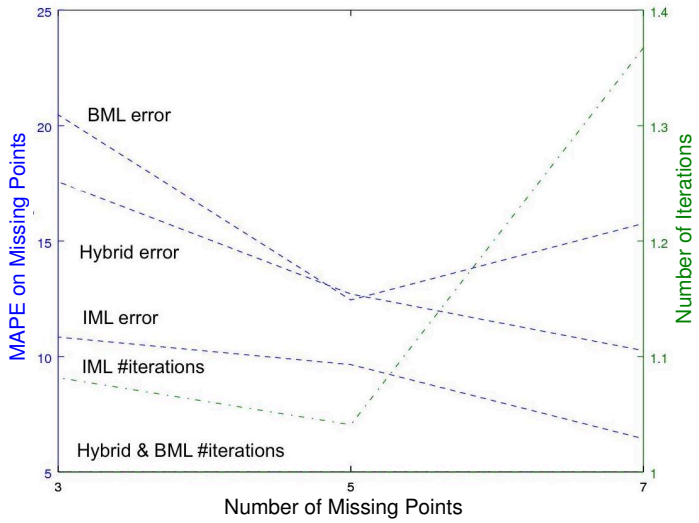


Figure 5.15 – Interpolation for Q40 query

(iii) seven points—20, 40, 48, 72, 80, 100, and 120 cores. Concerning the relative error, we observe in Figure 5.15 that the iterative ML approach gives better accuracy compared to our technique. Although the hybrid error for three missing points is high (17 %), in case of seven missing points the hybrid algorithm gets closer to iterative ML's performance. However, the number of iterations of the hybrid approach is smaller than the one of iterative ML, thus leading to lower costs, as reported in Figure 5.10.

5.5 CNN Models

In this section we report numerical results to support and validate the modeling techniques proposed for CNNs. In order to provide a reproducible experimental setting, we consider AlexNet [74], GoogLeNet [115], and VGG-16 [113] as CNNs, while the datasets are the ones released for ILSVRC2012.[6]

[6] http://www.image-net.org/challenges/LSVRC/2012/

Table 5.7
NVIDIA GPUs Specifications

Characteristic	M6000	P100	Unit
NVIDIA CUDA cores	3,072	3,584	-
GPU memory, DDR5	12	16	GB
Memory bandwidth	317	732	GB/s
Single precision operations	7.0	9.3	TFLOPS

We collected data from two computational nodes with different GPUs. The first has an Intel Xeon E5-2680 v2 2.80 GHz 10-core microprocessor, an NVIDIA Quadro M6000[7] GPU, and runs CentOS 6.8; the second sports an Intel Xeon E5-2680 v4 2.40 GHz 14-core CPU, an NVIDIA Tesla P100-PCIe[8] graphic card, and CentOS 7.4. Table 5.7 reports some relevant specifications of both GPU models. We report single precision operations per second since Caffe uses this precision by default in floating point arithmetics. Thanks to the increased speed and to the more than double memory bandwidth, P100 showed an improvement in end to end execution times in the range 40–90 %.

Exploiting an ad hoc benchmarking framework internally developed at IBM Research, we performed several runs of the three CNNs with varying batch sizes and iterations numbers. In every configuration we collected mainly two logs, one for the regular learning procedure and one for "timing" runs. Caffe provides this handy capability for obtaining detailed measurements of each layer composing the network: instead of proceeding with the usual workflow, which interleaves a number of training iterations with a streak of validation batches used only to assess the current classification quality, the framework focuses on recording the precise execution times for all the layers. We built the datasets for the per layer models by juxtaposing the average layer running times returned after Caffe timing runs and the complexity determined via the formulas in Table 3.1. On the other hand, we extracted from ordinary execution logs, via a custom parser, the time taken to perform both the training and testing phases of the CNNs, thus constructing datasets where these overall times are associated with the corresponding batch sizes and iterations.

As accuracy metric we consider signed relative errors, as defined in equation (5.1). Both the measured times T^r and the predictions T^s refer to the total time taken for CNN training. As additional accuracy metric, when we need to summarize the results, we take absolute values of the relative errors and compute the average, thus obtaining MAPEs. Our validation dataset consists of around 500 runs and in the following we report the most significant outcomes.

[7] https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/documents/75509_DS_NV_Quadro_M6000_US_NV_HR.pdf

[8] https://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-PCIe-datasheet.pdf

Table 5.8
Per Layer Model Validation, NVIDIA Quadro M6000

Network	Batch	Iterations	T^r [ms]	ε^1 [%]	ε^{GN} [%]
AlexNet	16	100	3,427	−0.71	−21.30
AlexNet	16	150	4,880	4.60	−17.10
AlexNet	16	230	7,488	4.52	−17.16
AlexNet	32	100	4,914	1.20	0.68
AlexNet	32	150	6,987	6.77	6.22
AlexNet	32	230	10,719	6.72	6.16
AlexNet	64	100	8,209	−1.16	15.10
AlexNet	64	150	11,555	5.34	22.65
AlexNet	64	230	17,790	4.91	22.16
GoogLeNet	16	100	7,666	2.93	2.93
GoogLeNet	16	150	10,823	9.36	9.36
GoogLeNet	16	230	16,578	9.47	9.47
GoogLeNet	32	100	13,201	−7.19	−7.19
GoogLeNet	32	150	18,874	−2.63	−2.63
GoogLeNet	32	230	28,542	−1.27	−1.27
GoogLeNet	64	100	24,299	−13.68	−13.68
GoogLeNet	64	150	34,088	−7.70	−7.70
GoogLeNet	64	230	52,468	−8.05	−8.05
VGG-16	8	100	18,440	−2.98	−15.96
VGG-16	8	150	26,157	2.60	−11.13
VGG-16	8	230	39,954	2.99	−10.79
VGG-16	16	100	33,283	−2.07	−9.41
VGG-16	16	150	46,621	4.87	−2.99
VGG-16	16	230	72,084	4.00	−3.79
VGG-16	32	100	62,875	−1.40	−5.43
VGG-16	32	150	88,732	4.81	0.52
VGG-16	32	230	136,685	4.32	0.06

5.5.1 Per Layer Model

Here we report the validation results of our proposed per layer modeling approach. The predictions computed via the models discussed in Section 3.5.1 are compared to the measured execution time for the relevant runs.

Table 5.8 lists the accuracy achieved on the M6000 node, alongside CNN, batch size, number of iterations, measured time T^r. For comparison, we report both the relative errors obtained by predicting with models learned on same CNN timing runs, in column ε^1, and the ones gotten when using the models for GoogLeNet across all the considered networks, in column ε^{GN}. In this way we underline the generalization capabilities of this approach.

Following a common practice, we varied the batch size according to a geometric progression of ratio 2, ranging from 16 to 64, except for VGG-16, which could not run at batch size 64 due to memory constraints: as reported in Table 3.2, it has the most activations, thus exhausting earlier the GPU's RAM. As far as accuracy is concerned, when using GoogLeNet's models errors in most cases remain below 20 % and the MAPE is 9.29 %. When considering CNN-specific models, instead, errors generally remain below 10 % and the MAPE settles at 4.75 %.

Another interesting aspect to highlight is that when the batch size in-

Table 5.9
End to End Model Validation, NVIDIA Tesla P100-PCIe

Network	$\tilde{b}$	$\tilde{\imath}$	n^{train}	n^{test}	MAPE [%]
AlexNet	16	150	6	140	3.18
AlexNet	24	170	12	140	6.43
AlexNet	32	200	20	140	3.01
GoogLeNet	16	150	6	140	5.79
GoogLeNet	24	170	12	140	1.09
GoogLeNet	32	200	20	140	2.43
VGG-16	16	150	6	140	3.70
VGG-16	24	170	12	140	2.20
VGG-16	32	200	20	140	3.41

creases, the CNNs show different efficiencies. When fixing the number of iterations and comparing T^{r} at various batch sizes, AlexNet has the most favorable ratio. We can interpret this result as an effect of the lower number of layers (see Table 3.2), which introduces fewer synchronization points between subsequent computation phases, thus less overhead.

5.5.2 End to End Model

This section describes the validation approach for the end to end model proposed in Section 3.5.2 and its results.

Recall that feasible batch sizes are limited by memory constraints, so in this experiment we vary b with step 8 to achieve a greater sample size: we ran experiments on the P100 node exploring the Cartesian product of $B = \{8, 16, 24, 32, 40, 48, 56, 64\}$ and $I = \{100, 120, 150, 170, 200, 230, 250, 300, 350, 400, 500, 700, 800, 950, 1000, 1100, 1400, 1600, 2000, 2300\}$.

In order to assess the quality of execution time prediction, for each CNN we split the dataset into training and test portions. Models were learned on the subset devoted to the training phase, while the final accuracy metric is the MAPE evaluated on the test set by comparing overall real execution times with the corresponding predictions. By splitting the dataset we can quantify the predictive capability on configurations not available during the learning via linear regression, particularly in terms of extrapolation towards larger batch sizes and more overall iterations. This approach to splitting data is motivated by the economic benefit of running only a limited number of small scale experiments, rather than spanning the whole domain where parameters can attain values.

Table 5.9 lists the results obtained with these experiments. The split between training and test set was performed by setting thresholds on the batch sizes and numbers of iterations allowed in the former. Every row corresponds to a model learned on the training set $\{(b, i) \in B \times I : b \leq \tilde{b}, i \leq \tilde{\imath}\}$, whose sample size is n^{train}. For ease of comparison, we used as test set only the subset of data points that do not appear in any of the training sets, hence each CNN is associated with a single test sample size n^{test}. All the three CNNs show good accuracy on the test set, with a worst case MAPE of 6.43 % even when rather small training sets are used.

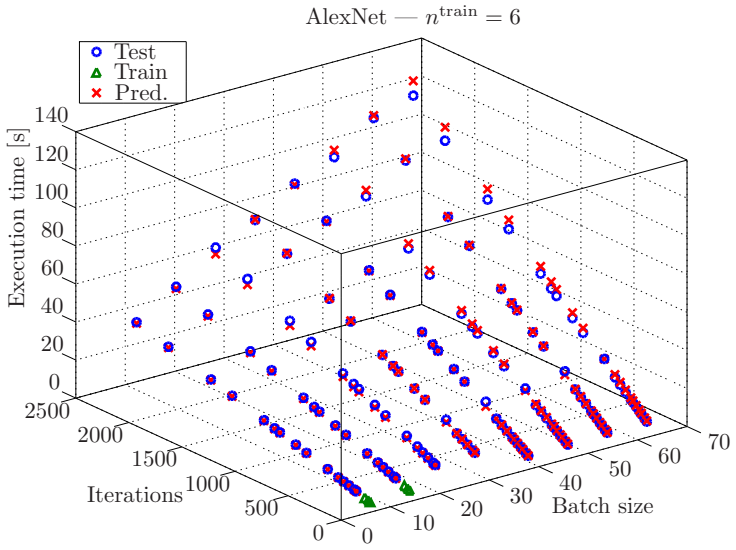


Figure 5.16 – AlexNet end to end time, $n^{\text{train}} = 6$

Figure 5.16 reports, as example, a plot depicting the results obtained with the model learned for AlexNet using the 6-element training set, which corresponds to the first row in Table 5.9. Triangles represent the data points in the training set, circles are real measurements in the test set, while crosses are the predictions given by the linear regression model, again only in the test set. The predicted execution times show a good accordance with the corresponding measurements, with a MAPE of 3.18 % on the test set. With this training set, the estimated coefficients are in the order of magnitude of 10^{-2} for $\hat{\beta}_i$ and $\hat{\beta}_b$, while $\hat{\beta}_{ib}$ is in the order of 10^{-3}. As the batch sizes b are typically between 10 and 100, the number of iterations i is typically larger than 10,000, the dominant term will be $i \cdot \hat{\beta}_i + i \cdot b \cdot \hat{\beta}_{ib}$. This outcome is quite intuitive, as the product $i \cdot b$ quantifies the total number of images fed into the CNN for processing: clearly the amount of input data has a major role in determining performance.

5.6 Discussion

The performance models described in Chapter 3 were evaluated here. All the simulation models achieve a good accuracy on average, with QNs settling at a 14.13 % relative error, SWNs at 9.08 %, and dagSim at 3.06 %. These good results motivate the choice to adopt them to predict performance during the optimization procedures presented in Chapter 4. In addition, these models are built on framework behaviors, so they are quite insensitive to the range where they are evaluated.

On the other hand, both hybrid and basic ML techniques offer models that are very fast to evaluate, given their algebraic closed form, but not as accurate. Most importantly, a general property of regression methods is their unreliability outside of the domain spanned by training data. At design time it is not possible to constrain an application to an already explored parameter range,

whence we apply ML only to provide the optimization procedures with a reasonable starting point for the search.

Alongside the already discussed validation results, this chapter also shows the evaluation of a deployment choice, whether or not to use spot instances, performed with SWNs. Thanks to the expressiveness of the formalism, it is possible to extend basic Hadoop or Spark models and add some complex behaviors, thus enabling the quantitative evaluation of various alternative design choices.

In the end, we also report the validation of two performance models obtained with linear regression to investigate CNNs running on GPGPUs. For this use case, simulations cannot be used due to the high level of parallelism attained in GPUs, which would lead to state space explosion and make simulation times impractical. However the presented ML approaches allow to attain very accurate models, with a MAPE of 9.29 % for the per layer method and always below 7 % in the end to end case.

Overall, the hereby presented experimental campaign quantitatively proves the accuracy of the proposed performance models, thus contributing to research question 1. Furthermore, Section 5.2.2 shows an example of using the proposed methods to assess the impact of spot instances, along the lines of research question 4. Adopting an expressive modeling formalism as SWNs, it is possible to investigate the trade-off between low operational costs and unreliability offered by the peculiar spot pricing model, which could make for an interesting design choice when SLAs are not too stringent.

CHAPTER 6

Optimization Techniques Validation

This chapter presents several validation analyses carried out to assess the quality of the solutions returned by the proposed optimization tools: D-SPACE-4Cloud, OPT_IC, and OPT_JR. To begin with, Section 6.1 uses profiling data about TPC-DS queries for the validation of D-SPACE4Cloud, with scenarios, real system experiments, and a scalability analysis. Then Section 6.2 presents a case study, where D-SPACE4Cloud is applied in the design process of a product developed by industrial partners. Similarly, Section 6.3 validates OPT_IC, the tool for determining optimal provisioning used in the EUBra-BIGSEA architecture, on an analogous application. At last, in Section 6.4 we report the validation study for the run time re-balancer tool, OPT_JR, while Section 6.5 concludes the chapter with a discussion of the achieved results, also in relation to research questions 2, 3 and 4.

6.1 Design Time Problem

In the following we show the results of several experiments performed to validate the approach proposed for the solution of the design time capacity allocation problem. All these analyses have been performed on two Ubuntu 14.04 VMs hosted on an Intel Xeon E5530 2.40 GHz equipped server. The first VM ran D-SPACE4Cloud and dagSim, with 8 vCPUs and 12 GB of RAM. The second one, instead, ran JMT 0.9.3, with 2 vCPUs and 2 GB of RAM.

6.1.1 Experimental Settings

Refer to Section 5.1 for details on the considered experimental platforms and the TPC-DS dataset generator, which was used to create the DB where to run queries. In particular, this section uses the profiles of the MapReduce R queries in Listing 5.1, as well as the Spark Q queries in Listings 5.2 and 5.3. On top of the VM types mentioned in Section 5.1, notice that this section also

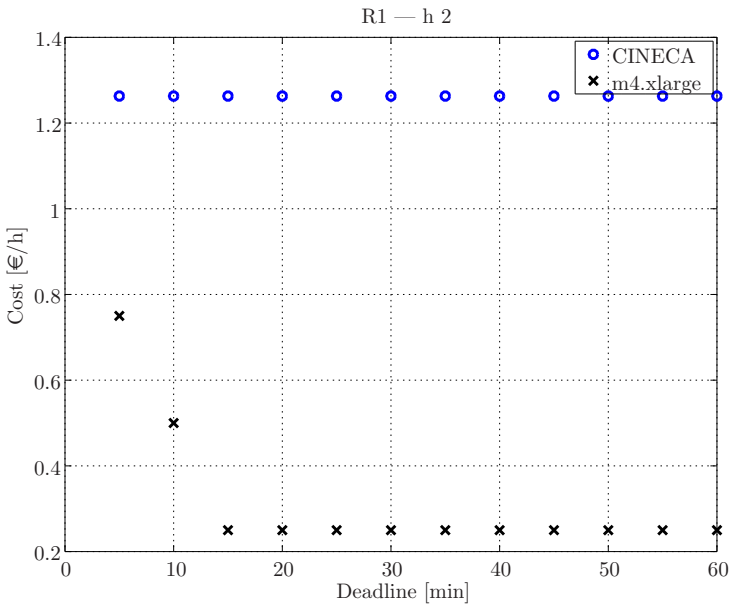


Figure 6.1 – Query R1, two concurrent users

considers deployments on Microsoft Azure based on A3, D13v2, and D14v2 instances.

Another considered deployment is an in house cluster based on IBM POWER8 (P8) processors available at Politecnico di Milano, which runs Red Hat Enterprise Linux 7.3 and Spark 1.4.1. The deployment includes four worker VMs with 11 cores and 60 GB RAM each, plus one similarly configured master node, for a total of 44 CPUs available for computation. Physical storage is provided by several fiber channel disks amounting to 12 TB. Spark executors were configured with 2 cores and 4 GB RAM, while 8 GB were allocated to the driver.

6.1.2 Scenario-based Analyses

The optimization approach described in Section 4.3 needs to be validated, ensuring that it is capable of catching realistic behaviors as one can reasonably expect of the system under analysis. We test this property with a set of assessment runs where we fix all the problem parameters but one and verify that the solutions follow an intuitive evolution.

The main axes governing performance in Hadoop or Spark clusters hosted on public clouds are the level of concurrency and the deadlines. In the first case, increasing h_i and fixing all the remaining parameters, we expect a need for more VMs to support the rising workload, thus leading to an increase of leasing costs. On the other hand, if at fixed parameters we tighten the deadlines D_i, again we should observe increased costs: the system will require a higher parallelism to shrink execution times, hence more containers to support it.

For the sake of clarity, in this section we performed single-class analyses: considering only one class ensures an easier interpretation of the results. Fig-

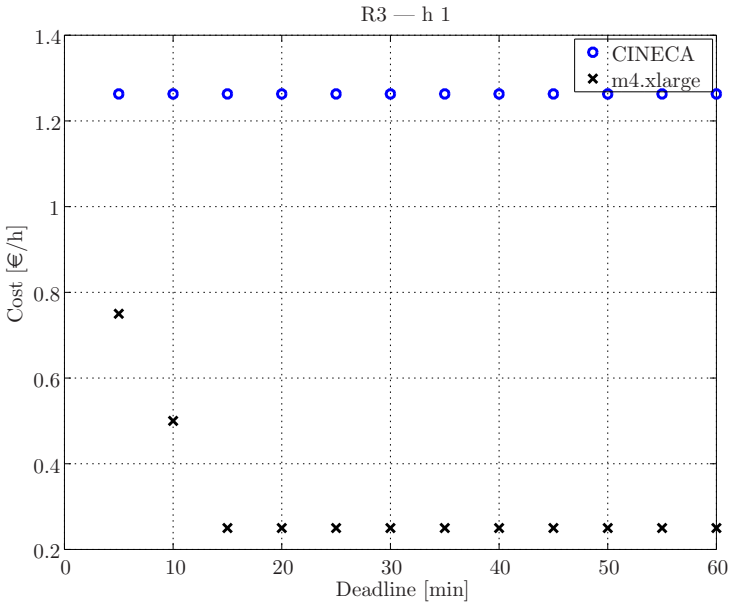


Figure 6.2 – Query R3, one concurrent user

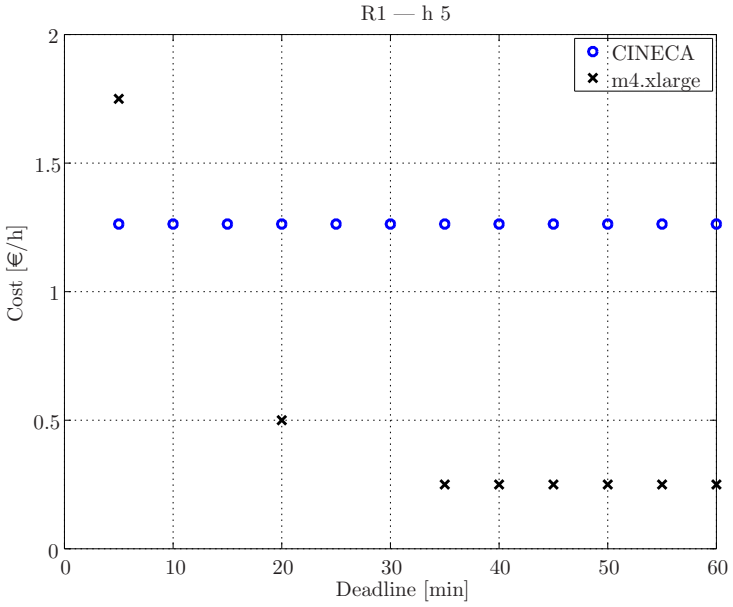


Figure 6.3 – Query R1, five concurrent users

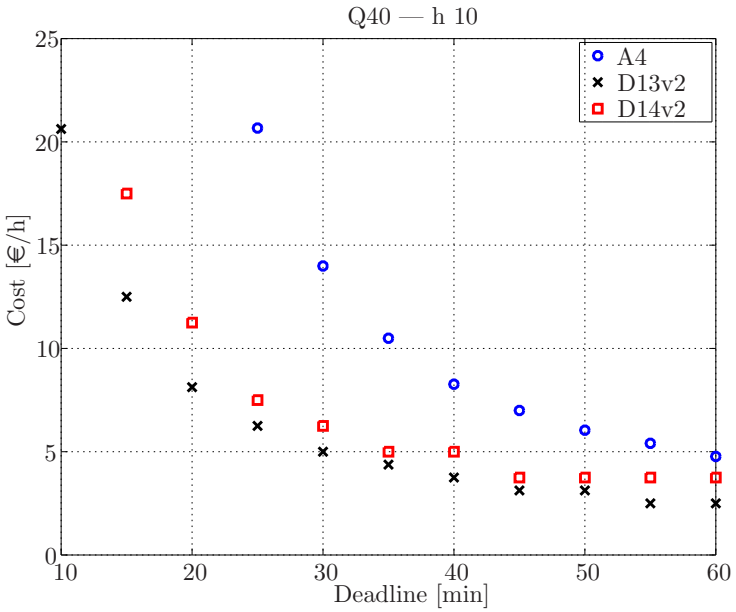


Figure 6.4 – Query Q40, ten users

ures 6.1, 6.2 and 6.3 report the solutions obtained with the 250 GB dataset on MapReduce queries when relying on the JMT simulator. The average running time for these instances is about two hours. All the mentioned figures show the cost in €/h plotted against decreasing deadlines in minutes for both the real VM types considered as candidate: CINECA is the 20-core node available on PICO, whilst m4.xlarge is the 4-core instance rented on Amazon AWS. In Figures 6.1 and 6.2 the expected cost increase due to tightening deadlines is apparent for both query R1 and R3. Further, in both cases it is cheaper to provision a cloud cluster consisting of the smaller Amazon-offered instances, independently of the deadlines. It is then interesting to observe that R1 shows a different behavior if the required concurrency level increases. Figure 6.3 shows that, as the deadlines tighten, it is possible to identify a region where executing the workload on larger VMs becomes more economic, with a 27.8 % saving.

Figures 6.4 and 6.5 show the behavior of several Spark runs on the 500 GB dataset. Q40 with ten users exhibits a straightforward behavior, with D13v2 instances always leading to cheaper deployments. In order to quantify monetary savings, we compute the ratio of the difference between costs over the second cheapest alternative. With this metric, D13v2 yields an average percentage saving around 23.1 % for Q40, hence this VM type proves to be the cheapest choice by a reasonable margin. On the other hand, a single-user Q52 provides a more varied scenario. As shown in Figure 6.6 for clarity, two VM types, namely, A3 and D12v2, alternate as the cheapest deployment when the deadline varies. By identifying the proper alternative, it is possible to obtain an average saving around 19.3 % over the considered range of deadlines. The maximum saving is about 36.4 %.

Overall, these results provide a strong point in favor of the optimization

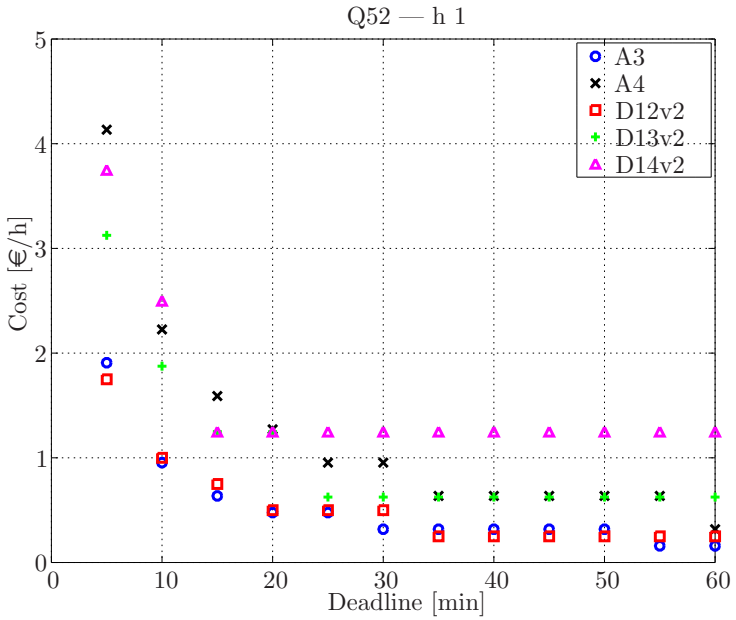


Figure 6.5 – Query Q52, single user

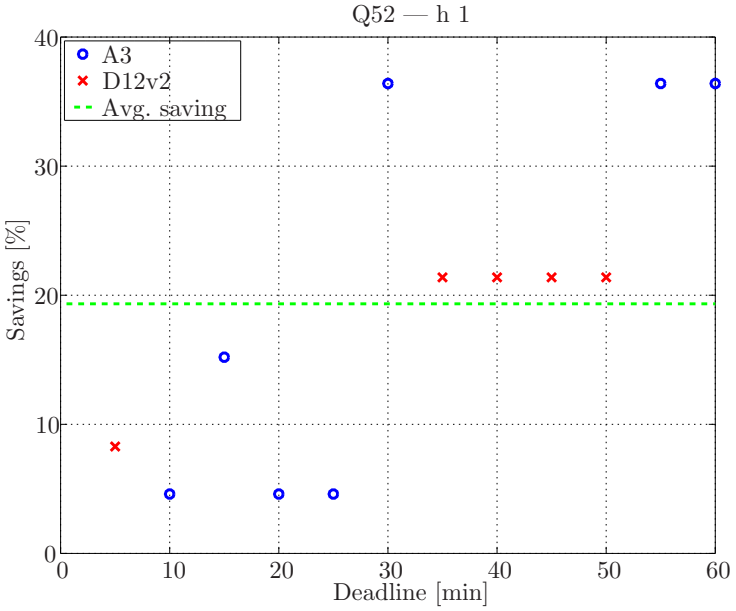


Figure 6.6 – Query Q52, single user, savings

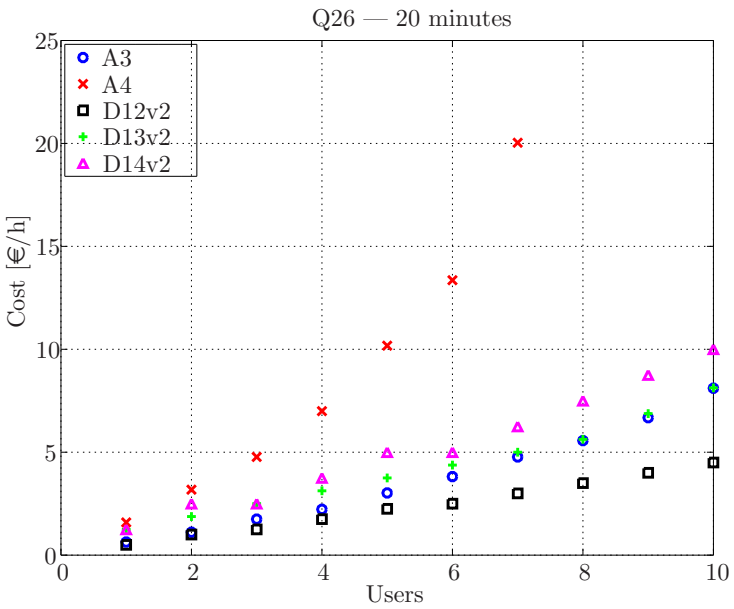


Figure 6.7 – Query Q26, multi-user

procedure implemented in our tool, as they prove that making the right choice for deployment can lead to substantial savings throughout the application life cycle. Take into account that Microsoft Azure suggests VMs of the D11–15v2 range for memory intensive applications, such as analytics on Spark or distributed DBs, whilst we showed that, depending on the workload, even the most basic offerings in the A0–4 range can satisfy QoS constraints with a competitive budget allocation. On top of this, D-SPACE4Cloud can also determine the optimal number of VMs to use in order to meet the QoS requirements, which is a nontrivial decision left to users.

In terms of execution times, D-SPACE4Cloud carried out the whole optimization procedure for Spark experiments within minutes. All the running times were in the range $[24, 560]$ s, with an average of 125.98 s. In these cases the search algorithm ran much faster due to the performance gain allowed by dagSim, which we used as simulator for the Q queries.

Figure 6.7 shows the results of a multi-user analysis over the 500 GB dataset. Fixing all the other parameters, in particular query Q26 and a deadline of 20 minutes, we varied the required concurrency level between 1 and 10 users with step 1. Here the D12v2 instances prove in every case the better choice, with an average 30.0 % saving in comparison to the second cheapest deployment.

In the end, to show the generality of the techniques implemented in D-SPACE4Cloud, we performed some runs of a random forest ML algorithm on the Azure platform. In particular, it was a binary classification problem on the logistic regression dataset from SparkBench [80], which comprises 10 millions entries with 100 features. Such an application has characteristics markedly different from query workloads, due to the iterative nature of its training algorithm. This analysis uses the obtained profile, a single user, and deadlines

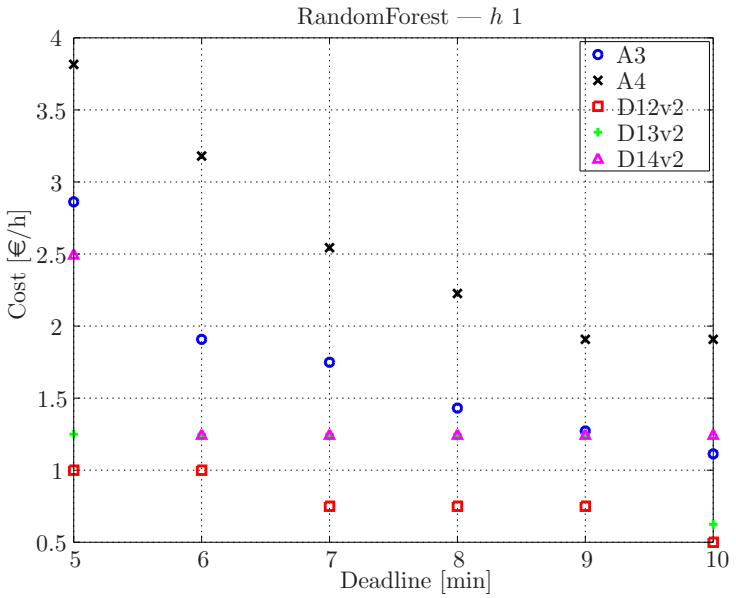


Figure 6.8 – Random forest, single user

varying between 5 and 10 minutes, with a 1-minute stride. As shown by Figure 6.8, in this case D12v2 instances make for the cheapest deployments irrespective of the deadline, with an average saving of 30 % in contrast to the second cheapest alternative, i.e., D13v2. Note that for 6–9 minutes both D13v2 and D14v2 enable same cost deployments.

6.1.3 Solution Validation on a Real Cluster

A further experiment was aimed at assessing the quality of the optimized solution obtained using D-SPACE4Cloud. Given a query and a deadline to meet, we focus on the execution time measured in a real cluster provisioned according to the number and type of VMs returned by D-SPACE4Cloud, quantifying the relative gap as a metric of the optimizer accuracy. Formally:

$$\varepsilon = \frac{D - T^{\mathrm{r}}}{T^{\mathrm{r}}} \tag{6.1}$$

where D is the deadline and T^{r} the execution time measured on the system, so that possible misses would yield a negative result.

We considered six cases, varying deadline and query, and ran D-SPACE4Cloud to determine the optimal resource assignment to meet the QoS constraints on D12v2 instances with a 500 GB scale factor. Table 6.1 collects data that relates to this experiment. Every row shows the relevant query and deadline, the optimal number of cores, the measured execution time, and the percentage gap ε. First of all, none of the considered runs led to a deadline miss. Moreover, the relative error is in most cases below 30 %, with a worst case result of 36.83 % and the average settling at 26.26 %. Overall we can conclude that the optimization tool is effective in identifying the minimum cost solution at design time, guaranteeing that deadlines are met as well.

Table 6.1
Optimizer single class validation, D12v2

Query	D [ms]	c	T^{r} [ms]	ε [%]
Q26	180,000	48	158,287	13.72
Q52	180,000	48	150,488	19.61
Q26	240,000	36	186,066	28.99
Q52	240,000	36	175,394	36.83
Q26	360,000	24	280,549	28.32
Q52	360,000	24	276,790	30.06

Table 6.2
Optimizer multi-class validation, D14v2

Query	D [ms]	c	T^{r} [ms]	ε [%]
Q26	720,000	16	533,731	34.90
Q40	720,000	16	530,122	35.82
Q52	720,000	16	562,625	27.97

In addition, we used D-SPACE4Cloud to optimize a multi-class instance on D14v2 VMs. Table 6.2 shows the results of this experiment, with three different queries subject to the same deadline that share a cluster of three worker nodes, for a total of 48 cores, under a work conserving scheduling policy. In this case, the worst case result is 35.82 %, with an average of 32.90 %. The accuracy is slightly lower than the single class scenario due to possible performance gains allowed by spare resources borrowed from other classes during their idle time, yet it remains good for practical applications.

6.1.4 Scalability Analysis

In this section we quantify the time taken to obtain the optimized solution for instances of increasing size, both in terms of number of classes and aggregate user count. All these runs exploited dagSim as simulator and Azure D14v2 VMs as target deployment.

The experiment considers three different queries, namely, Q26, Q40, and Q52, varying the deadline between five minutes and one hour with a five-minute stride, to obtain 12 sets of three distinct classes. Thus, we have Q26, Q40, and Q52 with deadline 60 minutes, then the three queries all with a deadline of 55 minutes, and so on. The instances are then created cumulatively joining the three-class sets following decreasing deadlines. For example, the configuration with 3 classes has $D = 60\,\mathrm{min}$, the second instance with 6 classes collects $D \in \{60, 55\}\,\mathrm{min}$, the third one adds $D = 50\,\mathrm{min}$, and so forth. We repeated this test instance generation with a required level of concurrency ranging from 1 to 10 users, but without ever mixing classes with different h_i: in any given instance, $\forall i \in \mathcal{C},\ h_i = \bar{h}$. In this way, we considered a total of 120 different test instances with varying number of classes and overall concurrent users: classes range between 3 and 36, while the aggregate number of users from 3 to 360.

Figure 6.9 shows the results of this experiment. The plot represents the mean execution time of D-SPACE4Cloud at every step. The number of users

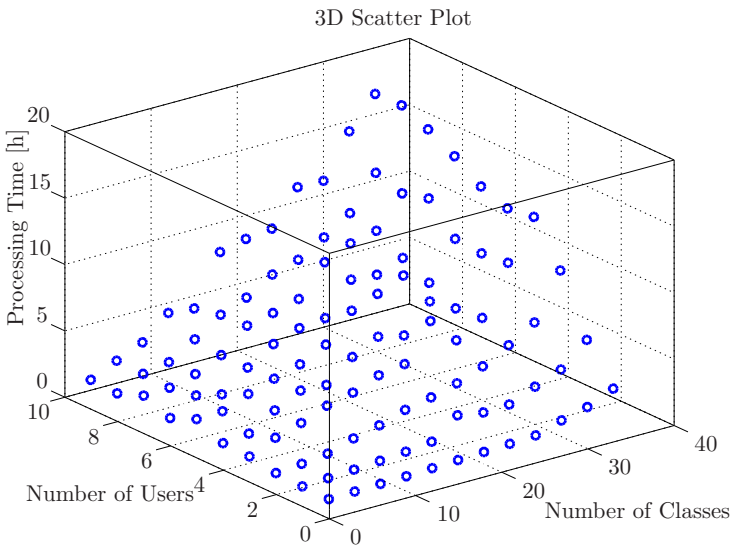


Figure 6.9 – Execution time for varying number of classes and users

is per class, thus, for example, the configuration $(24, 4)$ totals 96 users subdivided into 24 distinct classes with concurrency level 4. Overall, the average processing time ranges from around 40 minutes for three classes up to 12 hours for 36 classes. On the other hand, the best single instance, which needs only three minutes, is with three classes and one user each, while the longest running takes 16 hours and a half to optimize 36 classes with 9 users each.

The reported results show how D-SPACE4Cloud can solve the capacity allocation problem for cloud systems in less than one day, which is a reasonable time span for design time considerations. Furthermore, current best practices discourage hosting as many as 36 application classes and 360 users on a single shared cluster, hence several of the considered instances should be considered one order of magnitude larger than nowadays production environments.

6.1.5 Comparison with an Alternative Literature Proposal

As already highlighted in Chapter 2, currently in the literature there are no works studying the same problem solved by D-SPACE4Cloud. This notwithstanding, comparing the obtained results with a reasonable baseline is helpful in validating the quality of the proposed approach.

Hemingway [100] is not too far from the formulation used in D-SPACE4Cloud. The mentioned framework focuses on ML workloads and iteratively reaches, exploiting Bayesian optimization, the optimal configuration for recurrent jobs. For a job to be recurrent, it has to be the very same piece of software that runs repeatedly under the same deadline. Compared to the method discussed here, Hemingway has some shortcomings: i) it only considers single user scenarios, and ii) it must be applied fixing the application and SLAs.

In order to have a fair comparison, we take into account the underlying performance model, Ernest [121]. Since the model itself is valid only with a single user, this is the scenario considered in the following. The authors

Table 6.3
Optimal solution comparison, D12v2

Query	D [ms]	c^{r}	ε_{d} [%]	ε_{e} [%]
Q26	180,000	48	0.00	16.67
Q52	180,000	48	0.00	0.00
Q26	240,000	40	−10.00	0.00
Q52	240,000	40	−10.00	−10.00
Q26	360,000	24	0.00	16.67
Q52	360,000	24	0.00	0.00

propose to apply nonnegative least squares fitting and learn a performance model of the form:

$$t = \theta_0 + \theta_1 \frac{s}{m} + \theta_2 \log(m) + \theta_3 m, \tag{6.2}$$

where t is the execution time, s the input data size, and m the number of allocated machines. Each of the variable terms serves the purpose of modeling a specific common communication/computation pattern implemented in distributed applications: namely, the term in θ_1 relates to data-parallel processing, whilst the logarithmic term models tree-shaped aggregation, and $\theta_3 m$ highlights reduction operations targeting only one sink node.

Ernest's authors propose a methodology to train the model based on optimal experiment design, which, however, is out of scope in this dissertation. To have a fair comparison, the training datasets comprised all the runs done during the experimental campaign, which was very fine grained and included a very large set of experiments where the system configuration varied with two-core steps for every platform. In this way, model learning happens with an abundance of data, so as not to impair the baseline. We applied the learned model to predict performance across the whole range of possible core allocations, then obtained the optimum by inspection, taking the minimum cost configuration whose predicted time is feasible, i.e., shorter than the imposed deadline. Adopting this approach, we are sure to determine the global optimum associated with the baseline model. Similarly, we use measurements on the real system to determine by inspection the minimum cost configuration that meets a given deadline. The quality metric is then the signed relative error on the predicted allocation:

$$\varepsilon = \frac{c^{\mathrm{p}} - c^{\mathrm{r}}}{c^{\mathrm{r}}}, \tag{6.3}$$

where c^{p} is the optimal number of cores returned by D-SPACE4Cloud (respectively, Ernest) and c^{r} the actual optimum obtained by inspection from measurements on the real system. With this definition, when the optimizer assigns too scarce resources the error is negative.

Table 6.3 considers six cases, varying deadline and query, and lists the optimal resource assignment to meet the QoS constraints on D12v2 instances with a 500 GB scale factor. Every row shows the relevant query and deadline, the optimal number of cores, and the percentage errors with both alternatives. The two methods prove accurate, with relative errors that always remain below 20 %, on average 3.33 % for D-SPACE4Cloud and 7.22 % for Ernest.

Table 6.4
Optimal solution comparison, IBM POWER8

Query	D [ms]	c^{r}	ε_{d} [%]	ε_{e} [%]	$\varepsilon_{\mathrm{e}'}$ [%]
Q40	2,537,015	6	0.00	0.00	33.33
Q40	1,905,821	8	0.00	−25.00	25.00
Q40	1,566,992	10	0.00	−20.00	20.00
Q40	1,337,868	12	0.00	−16.67	16.67
Q40	1,165,666	14	0.00	−28.57	28.57
Q40	1,044,557	16	0.00	−25.00	25.00
Q40	964,752	18	0.00	−22.22	22.22
Q40	902,173	20	0.00	−30.00	10.00
Q40	854,692	22	0.00	−27.27	9.09
Q40	812,298	24	0.00	−33.33	8.33
Q40	784,445	26	−7.69	−38.46	0.00
Q40	763,447	28	−7.14	−35.71	0.00
Q40	743,283	30	−6.67	−40.00	−6.67
Q40	726,324	32	−6.25	−43.75	−6.25
Q40	718,641	34	−11.76	−47.06	−11.76
Q40	696,641	36	−5.56	−50.00	−16.67
Q40	687,299	38	−10.53	−47.37	−15.79
Q40	664,080	40	0.00	−50.00	−20.00
Q40	662,526	42	−4.76	−52.38	−23.81
Q40	663,823	44	−9.09	−54.55	−27.27

Table 6.4, on the other hand, reports an extensive comparison over the full range of experiments run on the internal cluster based on IBM P8 processors for query Q40. In particular, the table shows results obtained at a 1,000 GB scale factor, with deadlines set at the mean execution times (or response times [78]) measured on the real system. In this case, Ernest with the basic feature set widely underestimates the optimal resource allocation across the board. In order to obtain a fair comparison, the column marked $\varepsilon_{\mathrm{e}'}$ lists the errors yielded by the Ernest model tweaked using the additional term $\theta_4 s^2 m^{-1}$, at which the authors hint in their paper, suggesting that it is useful for some workloads. This analysis shows that the proposed approach exploiting D-SPACE4Cloud and dagSim attains a better accuracy than the baseline, with an average relative error of 3.47 % instead of the 16.32 % obtained by Ernest with the additional feature. The worst value for the error is also lower, −11.76 % against 33.33 %.

6.2 Data Privacy Case Study

Within the frame of DICE, Netfective Technology built a minimum viable product (MVP) to appraise the capabilities of big data in e-government applications, especially for tax fraud detection. Big data technologies have already proven how valuable they are to industries. Many businesses that have taken advantage of big data management and processing systems have increased their effectiveness and efficiency; whether it be for healthcare, advertising, or retail. Fraud recognition requires a holistic approach and a combined use of tactical or strategic methods and state of the art big data solutions. Tradi-

tional fraud detection practices have not been particularly successful largely because they come into play after the fact, whilst big data intelligence software can perceive the deviant behavior in real time, thereby enabling fiscal agencies to get better outcomes.

In the following we discuss a cost impact analysis carried out through D-SPACE4Cloud in order to assess the effects of various alternative data privacy techniques. This was done as part of the design process for NETF's MVP, Big Blu, which is aimed at supporting tax fraud detection thanks to big data technologies.

6.2.1 Background

Data anonymization, also known as de-identification, consists in techniques that can be applied to prohibit the recovery of information about individuals. For example, not allowing the result of a statistical query to be shown when the number of retrieved records falls below some threshold, or deliberately entering small inaccuracies or "noise" in the results of statistical queries makes the deduction of individual information more difficult.

In a DB, data is stored in tables and each record corresponds to one individual. Each record has a number of attributes, which are classified into three categories: i) key attributes, which uniquely identify individuals; ii) quasi-identifiers, which can be combined with external information to expose some individuals, or to reduce uncertainty about their identities; and iii) sensitive attributes, which contain sensitive information about individuals. In practice, each category requires different levels of privacy protection.

There are several techniques that can be applied to data before or along the big data process, in order to protect the privacy of individuals. Some of the most used are:

Generalization replaces quasi-identifiers for less specific, but semantically consistent, values. In this technique, a value is replaced for another more generic, yet faithful to the original.

Suppression deletes key identifiers or quasi-identifiers to form anonymized tables. It is used in the context of statistical DBs, which provide only summaries of the table data instead of individual data.

Encryption uses cryptographic schemes based on public or symmetric keys to replace sensitive data (key-attributes, quasi-identifiers, and sensitive attributes) for encrypted data. It transforms data and makes it unreadable to those who do not have access to the encryption key.

Perturbation (or masking) consists in replacing actual values for dummy data. The general idea is to randomly change data to mask sensitive information while preserving the critical structures inferred with modeling. Examples of masking techniques are: replacement, shuffling, blurring, redaction/nulling.

Hashing is the transformation of a string of characters into a, usually shorter, fixed length representation of the original string.

Big Blu, the prototype developed by NETF, sends an alert whenever a suspicious tax declaration enters the information system. Big Blu has been implemented by relying on open source state of the art big data frameworks such as Kafka—to manage high rate event flows, Cassandra—to store and query massive amounts of data, and Spark—to process huge volumes of data.

The application will be performing computation on the whole data, including newly generated inputs. This data has to be processed using fraud indicators, which are a set of rules prescribed by domain experts. In the case that a new fraud indicator is introduced, Big Blu has to proceed with a new batch processing phase on the entire DB. It must also be able to answer any query using a merge between old batch results and new real time computation. The result is a list of taxpayers who may be fraudulent.

6.2.2 Experimental Setup

In this section we present the cost impact of different privacy protection approaches. The first assessed privacy mechanism is masking. In our concrete case, we introduced a dictionary table, which implements a one-to-one mapping between clear and masked attributes. The other privacy mechanism considered in our evaluation is encryption using Advanced Encryption Standard (AES) with 256-bit keys. AES is a symmetric block cipher widely used to encrypt sensitive data. In our case, the IDs and sensitive data stored in the DB tables are encrypted and decryption is performed contextually while running queries. We use two sizes for the keys in order to better understand their performance overhead.

To evaluate the system performance against different privacy preserving mechanisms, we ran experiments on the Microsoft Azure platform with VMs of type D12v2—8 cores and 28 GB RAM, and D4v2—4 vCPUs and 28 GB memory. The number of nodes varied between 3 and 13, hence overall we performed experiments using up to 52 cores. Each query for each configuration was run 10 times to evaluate the average execution time.

In order to avoid any privacy and/or confidentiality issue, the discussed results were obtained working on a synthetic, but realistic, dataset. NETF developed a random taxpayer generator to fill the DB with millions of records, with a relational model designed so as to be realistic, generic, and neutral.

In the following, we consider three reference queries (see Listing 6.1), which will be the baseline for evaluating the performance and cost overhead introduced by the proposed privacy techniques. Query 1 accesses two tables to perform its analysis: Declare and TaxPayer. It intends to measure the difference between incomes earned by a taxpayer during two successive years. This is carried out to detect fraudsters by comparing the two incomes according to some criteria. For instance, if the income declared a certain year is less than 20 % (this percentage can be set as a parameter) of the one declared the previous year, then the taxpayer is flagged as suspect. Since incomes are stored in the table Declare, Query 1 executes two joins: the first to make sure that the two tax declarations relate to the same taxpayer; and the second to obtain the full set of information about them. The result of this query is saved and used by other queries by passing the expected arguments, such as the percentage of income decrease, and the number of years to take into account. Query 5 involves only the table Declare. This table contains all the information needed

```
select tp.id, tp.gender,
       tp.birthdate, tp.birthdepartment, tp.birthcommune,
       d1.taxdeclaration, d1.declarationdate, d1.income,
       d2.taxdeclaration as d2taxdeclaration,
       d2.declarationdate as d2declarationdate,
       d2.income as d2income
from Declare d1
inner join Declare d2 on d1.taxpayer = d2.taxpayer
inner join TaxPayer tp on d1.taxpayer = tp.id
```

(a) Query 1

```
select dic.und_id, tp.gender,
       tp.birthdate, tp.birthdepartment, tp.birthcommune,
       d1.taxdeclaration, d1.declarationdate, d1.income,
       d2.taxdeclaration as d2taxdeclaration,
       d2.declarationdate as d2declarationdate,
       d2.income as d2income
from Declare d1
inner join Declare d2 on d1.taxpayer = d2.taxpayer
inner join TaxPayer tp on d1.taxpayer = tp.id
inner join Dictionary dic on dic.id = tp.id
```

(b) Query 3

```
select *
from Declare d1
```

(c) Query 5

```
select tp.id, s.location, td.roomcount
from TaxDeclaration td, Signatory s, TaxPayer tp
where s.taxpayer = tp.id and s.taxdeclaration = td.id
```

(d) Query 7

Listing 6.1 – NETF case study reference queries

to know every individual income and other credentials helpful to justify the amount of tax that should be paid. Query 7 involves three tables: TaxPayer, TaxDeclaration, and Signatory. Each tax return must be signed by the taxpayers before they submit it to the fiscal agency. This query retrieves, among other things, the place of signature.

To achieve masking, we introduced a dictionary table that implements a one-to-one mapping between a clear ID and a masked ID. Query 3 is derived from Query 1. The idea behind Query 3 is to measure the overhead on the system due to the additional join required to read the clear IDs. Similarly, Query 6 and Query 8 are derived from Query 5 and Query 7, respectively, and are used to examine the overhead on the system due to the additional join. As previously discussed the second considered privacy mechanism is encryption with AES at 256 bit. In this case, the IDs and sensitive data stored in the Cassandra tables are encrypted and decryption is performed contextually while running Queries 1, 5 and 7. All the mentioned queries have been implemented in Spark 2.0.

We use the number of records in the TaxDeclaration table to express the dataset size. Experiments were performed in the range between 1 and 30 million records. In Kalwar et al. [70] one can find more details about the assessment of performance degradation due to the use of privacy preserving

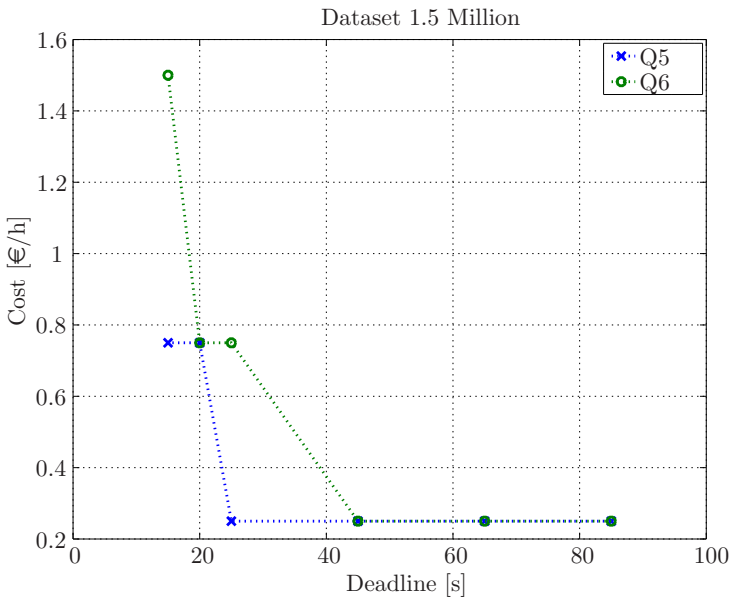


Figure 6.10 – Cost impact of masking, Queries 5 and 6, 1.5-million dataset

techniques, but here we only discuss the evaluation of their cost impact, which was performed via D-SPACE4Cloud. In particular, we investigated this aspect only in the configurations that showed the most degraded performance, with the rationale of assessing the worst case scenarios.

6.2.3 Cost Impact Analysis

In case of performance degradation, we used D-SPACE4Cloud to evaluate the cost impact of the implementation of privacy mechanisms when the two versions of the same query (i.e., plain and anonymized) need to be executed within the same deadline. The initial deadline was set according to the time measured on the real system with the smallest configuration. The deadline is decreased iteratively with a step whose range is $[5, 500]$ seconds: the difference was not fixed and varied according to the dataset size and privacy technique being considered. In this way, multiple optimization instances are considered for each experiment.

Initially we considered Query 5 and 6 with the 1.5-million dataset and an 85 s initial deadline. Then, the deadline was iteratively decreased by 20 s in order to consider 10 optimization instances. The results are reported in Figure 6.10. From the experimental results one can see that above 45 s and for 20 s no extra costs are incurred, while for the 25 s and 15 s deadlines the cost overhead due to the masking technique is between 50 and 66 %. Deadlines lower than 15 s resulted too strict and D-SPACE4Cloud did not find any feasible solution.

Figure 6.11 reports the results for Queries 1 and 3, when the 10 millions entries dataset is considered. The initial deadline is set to 3,500 s, that is the maximum execution time registered on the real cluster for both queries, and is iteratively reduced by 500 s. The results show that the cost overhead due to

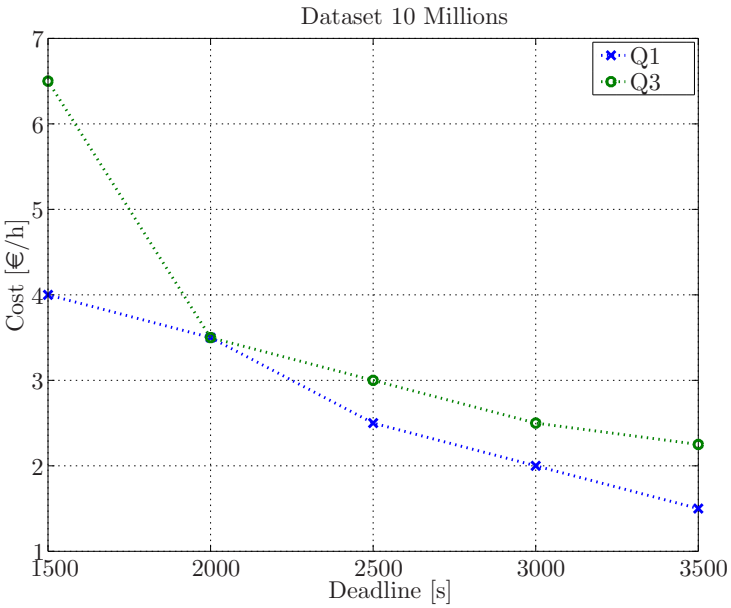


Figure 6.11 – Cost impact of masking, Queries 1 and 3, 10-million dataset

the masking technique is between 33 and 40 % and no extra costs are incurred for deadlines larger than 2,500 s. Deadlines lower than 1,500 s were too strict and no feasible solutions could be found.

Further experiments were targeted at encryption. We selected the 10-million dataset and evaluated Query 7, encrypted with AES 256 bit and unencrypted, for which we registered the largest performance degradation, about 5 %. In this way the results we achieve will be conservative. The initial deadline was set equal to 80 s, which was then iteratively reduced by 5 s at a time. The results reported in Figure 6.12 show that a 50 % cost overhead is achieved at 40 s, which is also the minimum deadline that can be feasibly supported by the system.

Finally, we report the results we achieved by considering the largest dataset, i.e., 30 millions, for Query 5. The initial deadline was set to 80 s, which then was iteratively reduced by 20 s. Figure 6.13 reports the cost for AES 256 bit encryption. The experiment shows that the additional cost due to encryption is only 13 % at maximum, while below 40 seconds D-SPACE4Cloud could not find any feasible solution.

From these results, we cannot state clearly which approach leads to higher cost impacts, since they alternate as cheapest choice depending on the query at hand and the considered dataset size. However this proves the worth of D-SPACE4Cloud, as mapping performance degradation to cost overhead is not a straightforward task to accomplish.

6.3 OPT_IC Case Study Validation

As noted in Section 4.4, the OPT_IC component developed in EUBra-BIGSEA substantially solves a formulation of problem (P1), which is presented in Sec-

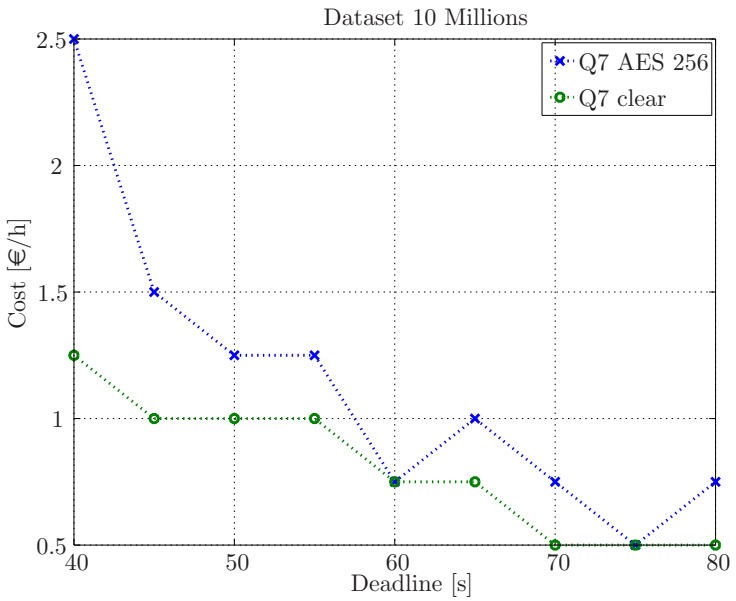


Figure 6.12 – Cost impact of encryption, Query 7, 10-million dataset

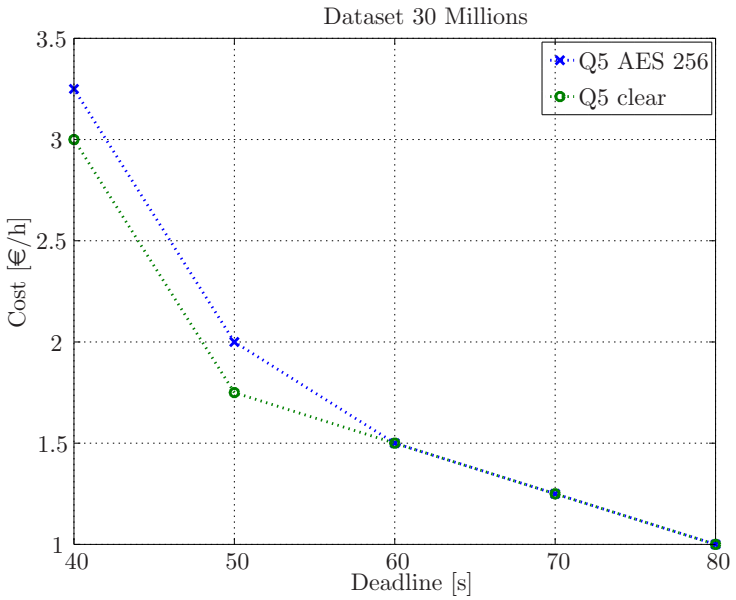


Figure 6.13 – Cost impact of encryption, Query 5, 30-million dataset

tion 4.3, where the supporting VM type is fixed. Here we show the validation of OPT_IC with an example application developed by Brazilian research partners: BULMA.

The problem faced by BULMA is identifying bus trajectories from sequences of noisy geospatial and temporal data sources, also known as map matching problem. It consists in performing the linkage between buses' GPS trajectories and the corresponding road segments, i.e., predefined trajectories or shapes, on a digital map. In this sense, BULMA is a novel unsupervised technique capable of matching a bus's trajectory with the "correct" shape, considering the cases in which there are multiple shapes for the same route, which is a common event in many Brazilian cities, like Curitiba and São Paulo. Furthermore, BULMA is able to detect deviations from the prescribed trajectories and mark them in its output. The goal of BULMA is to provide high quality integrated geospatial and temporal training data to support predictive ML algorithms of intelligent transport systems applications and services.

There are two types of files used by BULMA, namely, shape and GPS files. The first consists in data for the same root and corresponds to a rich/detailed set of georeferenced points describing the trajectory that a bus should follow, i.e., the predefined bus trajectory. It is extracted from General Transit Feed Specification data, which is an input file released by an operator or authority and containing details about public transit supply. Regarding GPS files, they contain all the GPS trajectories of a city's bus fleets during a certain period of time. In our case, each GPS file corresponds to all the GPS trajectories of the city's bus fleet during a day. Thus, BULMA matches a bus trajectory with the "correct" shape, considering the cases in which there are multiple shapes for the same route. As an example, Figure 6.14 includes two shapes describing the trajectories that a bus may follow to serve route 022 in the same direction. Most state of the art techniques are able to detect whether or not a bus is performing route 022, but, to the best of our knowledge, none of them is able to indicate if a bus will start its trajectory from point A or B. This is relevant when feeding the matched trajectory into a predictive algorithm for the purpose of arrival time estimation: if the generated trajectory is erroneous, so will be the output arrival time, too. Now, in Figure 6.14 the correct starting point of the prescribed trajectory is A, hence running any predictive algorithm on the matched trajectory starting from B would yield a wrong estimation of the arrival time at C. For all the aforementioned reasons, BULMA solves this problem with a good accuracy.

BULMA has been implemented in Spark and we performed the experiments by running the application in the Microsoft Azure HDInsight platform, considering Spark 2.1.0. Workers were deployed on D4v2 VMs, with 8 cores and 28 GB of memory, and every executor was configured to run with 8 cores and 16 GB of RAM. To check the performance of our tools, we considered a set of different configurations with a varying number of nodes in the cluster. In particular, we performed runs from 1 to 6 nodes and generated 5 days' worth of GPS training data. To validate OPT_IC we aimed at assessing the quality of the returned optimal solution. Assuming to optimize the initial deployment of BULMA under a deadline constraint, we focus on the response time measured in a real cluster provisioned according to the number of VMs determined by the optimization procedure, using the relative gap as a metric of the optimizer accuracy, as in (6.1).

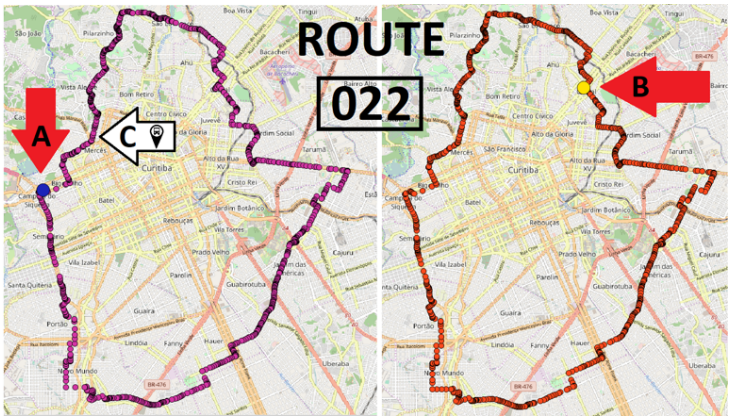


Figure 6.14 – Two different shapes describing the trajectories a bus may follow to serve route 022

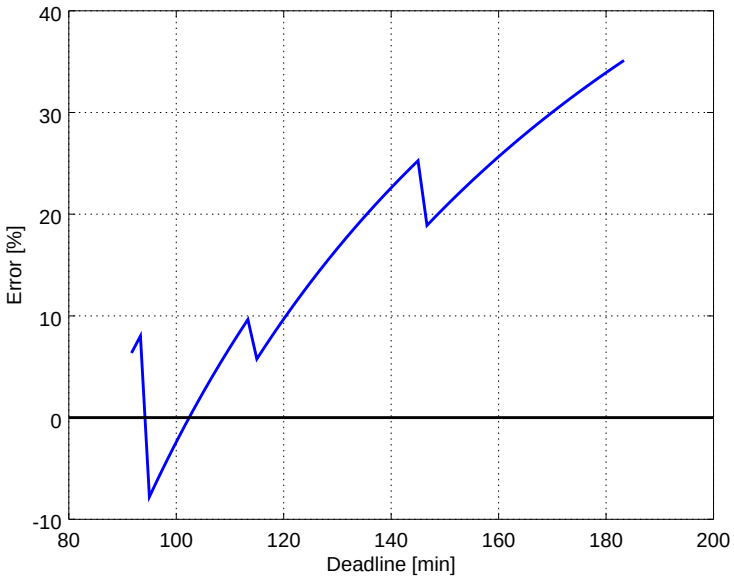


Figure 6.15 – OPT_IC percentage error as a function of the deadline

```
select i_brand_id brand_id, i_brand brand, sum(ss_ext_sales_price) ext_price
from date_dim, store_sales, item
where date_dim.d_date_sk = store_sales.ss_sold_date_sk
  and store_sales.ss_item_sk = item.i_item_sk
  and i_manager_id = 36
  and d_moy = 12
  and d_year = 2001
group by i_brand, i_brand_id
order by ext_price desc, i_brand_id
limit 100
```

Listing 6.2 – TPC-DS query Q55

We considered 56 cases, varying the deadline between 5,500 s and 11,000 s with step 100 s, and ran OPT_IC to determine the optimal resource assignment to meet the QoS constraints on D4v2 instances. Figure 6.15 plots the data we obtained. We experienced a deadline miss only in 5 out of 56 cases, amounting to the 8.9 %. Moreover, the relative error is always below 35 %, with a worst case result of 35.12 % and the average of the errors taken in absolute values settling at 18.13 %. From the plot, we observe an abnormal behavior, i.e, some rapid jumps, which are related to changes in the deployment configuration suggested by OPT_IC, specifically the number of required VMs. In particular, at most the gap in terms of required VMs is 1 and the accuracy increases when the deadline is tight, i.e., for larger clusters. Since the initial deployment can also be updated by the proactive policies module, overall we can conclude that the OPT_IC tool is effective in identifying the minimum cost initial deployment, guaranteeing that deadlines are met as well.

6.4 OPT_JR Validation

The aim of this section is the analysis of the accuracy and scalability of OPT_JR algorithm when used in the deployment of real systems. Consistently with what detailed in Section 5.1, also here we use synthetic DBs created with the TPC-DS dataset generator at scale factors of either 500 GB or 1,000 GB. The considered queries are Q26, Q40, and Q52—listed in Listings 5.2 and 5.3, as well as a fourth query, Q55, see Listing 6.2. Also in these scenarios, since application profiles have a statistical nature, we collected ten runs per query on each platform.

Real data for these analyses comes from two deployments. One is a Microsoft Azure HDInsight[1] cluster based on D13v2 VMs. Each VM boasts 8 cores, 56 GB of RAM, and 400 GB of local SSD, so they could support Spark executors with 8 cores and 40 GB RAM. The cluster was composed of two master nodes dedicated to 4 GB Spark drivers and six workers that hosted executors, for an aggregate 48 vCPUs. Section 6.1.1 details the second deployment based on IBM P8 processors.

The re-balancer tool, OPT_JR, has been executed on a server with two Intel Xeon Silver 4114 2.20 GHz CPUs and 48 GB of RAM.

[1] https://azure.microsoft.com/en-us/services/hdinsight/

Table 6.5
OPT_JR model validation Test 1, all weights set to 1

Query	M	m	V	v	D [s]
Q26	28	8	4	2	1,000
Q52	28	8	4	2	1,000
Q40	56	18	4	2	1,000
Q55	56	18	4	2	1,000

Table 6.6
Differences of OPT_JR validation tests with respect to Test 1

Test	N (500 GB)	N (1,000 GB)	p	w_{Q52}
Test 1	24	44	1.33	1
Test 2	48	88	0.66	1
Test 3	16	28	2.00	1
Test 4	24	44	1.33	10

Table 6.7
OPT_JR model validation, Test 5

Query	M	m	V	v	D [s]
Q40	56	18	4	2	600
Q55	56	18	4	2	800
Q26	28	8	4	2	800
Q40	56	18	4	2	800
Q52	28	8	4	2	800
Q55	56	18	4	2	800
Q26	28	8	4	2	1,000
Q40	56	18	4	2	1,000
Q52	28	8	4	2	1,000
Q55	56	18	4	2	1,000

6.4.1 Scenario-based Analyses

As a first form of validation, we used data profiled on the P8 cluster to handcraft eight test cases, whose goal is investigating the impact of various problem parameters on the performance and effectiveness of OPT_JR. In every case the maximum number of iterations allowed to OPT_JR is set to 10. Each test is characterized by a pressure p, defined as:

$$p = \frac{\hat{c}}{N}, \tag{6.4}$$

where $\hat{c}$ is the minimum number of cores that does not cause a deadline violation, as obtained via OPT_IC, and N is the number of available cores.

Table 6.5 describes Test 1, which is the base test and includes four queries. Table 6.6 details the differences of Tests 2, 3, and 4 with respect to Test 1. In Test 2, N is increased to get a pressure equal to 0.66: there are more available cores than required. On the contrary, in Test 3, N is decremented to set the pressure equal to 2.00: the number of available cores is half of the required. In Test 4 the weight of Q52 is set to 10 in order to show how the algorithm

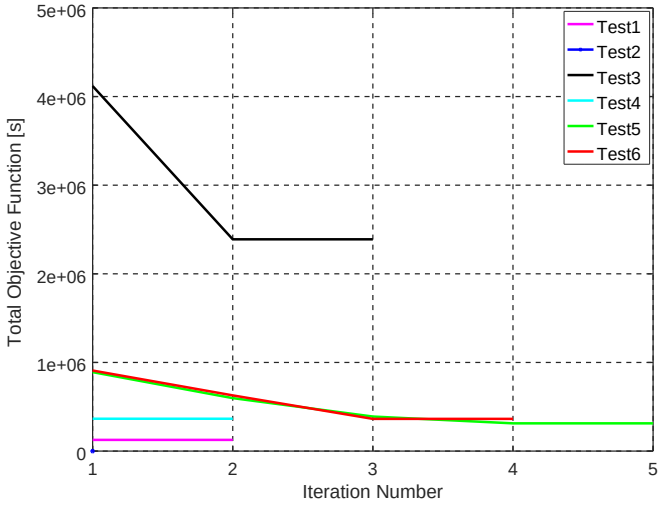


Figure 6.16 – OPT_JR validation (1,000 GB): overall weighted tardiness vs. iterations number

actually considers weights in computing the solution, but the other parameters are the same as in Test 1. On the contrary, Test 5 is more complex, with details in Table 6.7: multiple instances of the same query are considered so that the overall number of running applications is 10. We repeated the above described tests both with a 500 GB scale factor and at 1,000 GB. Finally, Test 6 uses the same applications of Test 5 with both datasets at the same time, thus resulting in two replicas of all the queries in Table 6.7. N has been set to 78 in the 500 GB instance of Test 5, then to 150 for 1,000 GB, and 226 in Test 6.

In the following we discuss the results we achieved. Comparing the parameters and the results for Tests 1, 2, and 3, it is possible to see that, by reducing the number of available cores, both the number of iterations (see, e.g., Figure 6.16) and the execution time (see Table 6.9) tend to increase. This was expected, since the smaller number of available cores in Test 3 makes the problem more difficult to solve. For Test 2 a single point is reported: in the first iteration, the objective function is always 0 since all the queries meet the soft deadline constraint.

The curves reporting the values of the total weighted tardiness, in Figure 6.16, decrease progressively on each subsequent iteration, thus proving that the local search technique is able to improve the initial heuristic assignment by selecting the best core configuration swap at each iteration. In all the tests the algorithm stops before reaching the maximum number of iterations, experimentally set to 10, since the candidates set U becomes empty, demonstrating that the optimization algorithm identifies in a few iterations the final local optimum solution. Note that final iterations never apply any change, consistently with Algorithm 4.5.1, hence the objective function remains unchanged in correspondence of each test's last step.

Finally, the results about the cardinality of the candidates list U are presented in Figure 6.17. This parameter plays a crucial role to understand algo-

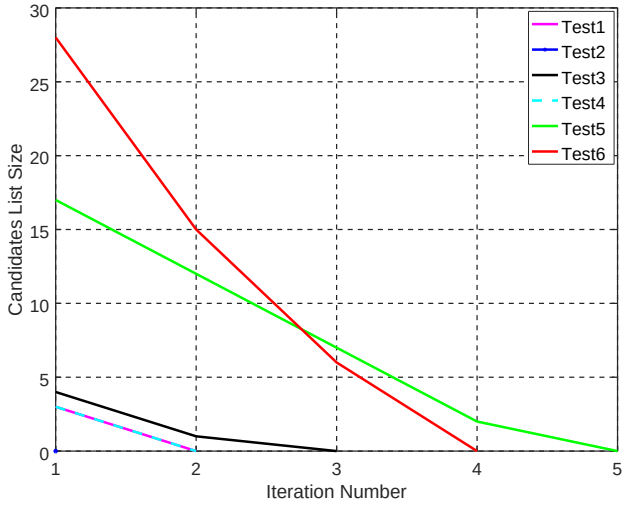


Figure 6.17 – OPT_JR validation: cardinality of candidates list U vs. iterations number

Table 6.8
Number of cores assigned to Q52, 1,000 GB dataset

Test	c
Test1	8
Test2	12
Test3	4
Test4	12

rithm performance, since this list's size corresponds to the number of times the predictor is invoked: a time consuming operation, since it implies, among other operations, a call to the Lundstrom tool as an external process. Moreover the size of set U has a monotonic decreasing behavior with respect to the number of elapsed iterations. Also in this plot only one point is reported for Test 2: since the initial solution already meets all the deadlines, no further change gets evaluated.

Table 6.8 shows the different number of cores assigned to Q52 in different tests when the dataset size is set to 1,000 GB and how the different parameters can impact on the produced solution. The minimum number of cores necessary to satisfy the deadline of 1,000 s is 12. In Test 1, since the number of available cores is less than the requirement—p is 1.33, not all the needed resources can be assigned. For this reason, in the optimal solution only 8 cores are assigned to Q52. On the contrary, Test 2 is characterized by $p < 1$, i.e., there are more resources than required, so all the necessary cores can be given to Q52. Test 3 is characterized by $p = 2.00$: for this reason, the number of cores for Q52 is even lower than in Test 1, with 4 instead of 8. Finally, in Test 4 there are not enough resources to satisfy all the requirements, just as in Test 1, yet Q52 has a larger weight than the other queries and it receives all

Table 6.9
OPT_JR execution times and speedup with different numbers of threads

Test	f [GB]	$T^{o}(1)$[s]	$T^{o}(2)$[s]	$S(2)$	$T^{o}(4)$[s]	$S(4)$
Test 1	500	13.73	12.18	1.13	8.86	1.55
Test 1	1,000	34.86	25.61	1.36	20.31	1.72
Test 2	500	13.88	11.64	1.19	8.96	1.55
Test 2	1,000	19.21	14.34	1.34	11.34	1.69
Test 3	500	13.78	11.65	1.18	8.83	1.56
Test 3	1,000	45.70	34.91	1.31	29.43	1.55
Test 4	500	13.77	11.62	1.19	9.33	1.48
Test 4	1,000	34.09	25.66	1.33	20.33	1.68
Test 5	500	60.86	48.99	1.24	42.23	1.44
Test 5	1,000	224.89	160.75	1.40	128.40	1.75
Test 6	both	313.04	229.82	1.36	181.10	1.73

Table 6.10
Description of the case study scenario

Query	f [GB]	w	t^{s} [s]	D [s]
Q20	1,000	5	0	780
Q52	1,000	1	320	600
Q55	1,000	1	320	600

the 12 required cores.

6.4.2 Performance Evaluation

OPT_JR exploits a set of OpenMP[2] directives to solve the re-balancing problem in multi-threaded mode. The objective of this approach consists in exploring the neighborhood with a parallel process, in such a way that the Lundstrom performance predictor gets invoked concurrently across multiple candidates.

Table 6.9 shows the performance speedup obtained on Tests 1–5, with both dataset scale factors 500 GB and 1,000 GB, as well as on Test 6. Specifically, the tests were executed with the Lundstrom predictor in single and multi-threaded mode, using either 2 or 4 cores. The usage of more than 4 CPUs does not provide significant advantages on these tests, because of the limited size of the candidate moves list U (see Figure 6.17). Both the optimization time, T^{o}, and the speedup, S, are expressed as functions of the number of threads. The above results suggests that our approach is adequate to manage at run time real clusters supporting long running batch queries, as the time requested to obtain a new configuration is in the order of a few minutes.

6.4.3 Real Scenario Case Study

Here we present the behavior of OPT_IC and OPT_JR in a real use case. This scenario considers three queries, i.e., Q20, Q52, and Q55, submitted for execution on the dataset of scale factor 1,000 GB over a cluster with 6 D13v2

[2] http://www.openmp.org

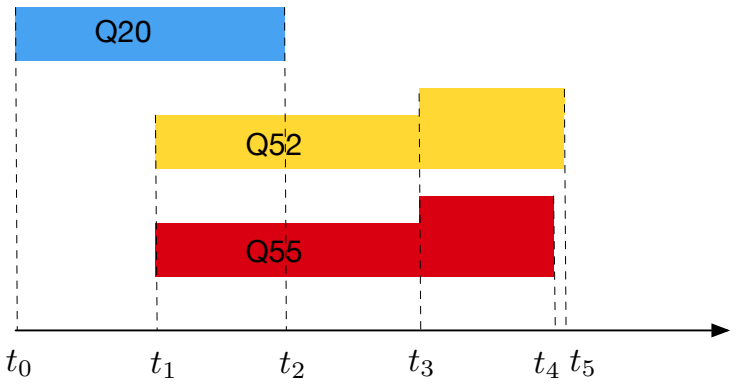


Figure 6.18 – Usage of resources of queries run in the case study

VMs, for a total of $N = 48$ cores. The three submissions happen in different instants. All the queries have been initially executed separately to collect the execution logs fed into dagSim and Lundstrom.

The details of the scenario are presented in Table 6.10: for each query there is the dataset scale factor, weight, submission time, and deadline. At t_0 only Q20 is submitted for execution on the cluster, with its deadline set to 780 s. OPT_IC estimates that 16 cores, which correspond to two VMs, are needed to fulfill the deadline. Since the full cluster is initially available, all the required VMs can be assigned to Q20. After about 320 s, at t_1, Q52 and Q55 are submitted with both deadlines set to 600 s. OPT_IC estimates optimal allocations of 30 and 28 cores, respectively, which can be supported by 4 VMs per query. Since the cluster does not have enough spare capacity, we are in heavy load condition. OPT_JR is triggered to minimize the weighted tardiness by re-balancing VMs across queries: in the identified solution two VMs are assigned to each query. It is worth noting that with this configuration Q20, which has the largest weight, is expected to meet its deadline, while Q52 and Q55 are not.

After about 250 s, t_2, Q20 ends its execution. Nevertheless, we are still in heavy load condition: the execution of Q52 and Q55 was slowed down by the limited number of cores, so each query would require a larger allocation than initially estimated. For this reason OPT_JR is executed again to re-balance resources between the two. Both tardiness values are similar, so OPT_JR provides each with half the available resources, that is 24 cores or 3 VMs. The detection of the end of Q20 and the successive invocation of OPT_JR takes only some seconds, but the implementation of the suggested solution takes some minutes, due to booting time and Spark configuration reload. At time t_3, 340 s later, the new resources are actually available to Q52 and Q55, which are supported by three VMs each from now on. When Q55 ends, t_4, Q52 is still running. Nevertheless, its end is foreseen in a few seconds: for this reason its number of executors is not incremented, since the overhead due to rescaling can nullify the benefits of a resource increment. In the end, in a few seconds also Q52 ends and the presented scenario completes.

To evaluate the accuracy of OPT_JR, we compare the overall weighted tardiness computed at each invocation with the actual value. At t_1, only Q52 and

Q55 contribute to the objective function, since Q20 is expected to end before its deadline. The estimated objective function is 908 s, while the actual value is 924 s. The relative error is very low, only 1.73 %, thanks to the good accuracy of the simulator. It is worth noting that, in the computation of both the estimated and real objective function, we are assuming that Q52 and Q55 will be always executed with the resources allocated to them at the moment. Since more resources will then be made available, the final aggregate tardiness will be smaller.

The second invocation of OPT_JR occurs at t_2: the estimated tardiness value is 320 s, while the actual value is 406 s, for a relative error of 21 %. The larger error is mainly due to the delay in assigning new resources. OPT_JR is assuming that they are available at t_2, yet they were ready only at t_3. This delay causes also a delay in the end of the execution of Q52 and Q55, resulting in a greater tardiness for both. Nevertheless the use of OPT_IC and of OPT_JR allows the user to satisfy Q20's deadline, thanks to higher weight, and to minimize tardiness for the others.

6.5 Discussion

The present chapter validates under several perspectives the optimization techniques described in Chapter 4.

Initially we study the effect of several parameters, such as deadlines and concurrency levels, on the cost to run a cluster that supports some DIAs. Our analyses suggest, in particular, that at times even VM types not optimized for a given kind of workload may become the best choice. More generally, taking such a decision with the help of the proposed performance modeling and optimization approach enables savings up to 36.4 %. These results, highlighting situations where the initial deployment choice inspired by framework characteristics is suboptimal, strongly provide an answer to research question 3.

D-SPACE4Cloud ensures that deadlines are met, with errors in comparison to the measured execution times below 30 %, and provides optimal solutions within hours even when dealing with extremely large instances, thus guaranteeing solution times absolutely acceptable during the design phase. In addition to these general analyses, we also applied D-SPACE4Cloud to case studies for the design of a real MVP: the tool predicted the effect on costs of a set of alternative design choices for an application centered on privacy. In particular this case study, presented in Section 6.2, shows how the devised optimization procedures can valuably be exploited in real world projects for the exploration of various architectural choices, thus giving an important point in favor of research question 4.

Afterwards, we also provide validation results for the run time aspects studied in the frame of EUBra-BIGSEA. We begin by assessing a number of test cases devised to highlight the capability of OPT_JR to address variations in the problem instance. In line with expectations, the tool assigns more or less resources to each considered application depending on the pressure and contention in the system. On top of this, we showed how the optimization method can provide a minimum weighted tardiness solution within minutes, even when the submissions become numerous. At last, by enforcing on a real

system the optimal solution returned by OPT_JR, we assessed its quality and quantified a 21 % relative error.

Overall, the relative errors listed in the current chapter validate the fact that the proposed optimization techniques are accurate, thus satisfying research question 2. Even at design time, before actually deploying the system, it is possible to predict with fair confidence the final behavior and to determine the optimal configuration required to meet QoS constraints.

CHAPTER 7

Conclusions and Future Work

Throughout this dissertation we presented several techniques to address a number of related problems of interest in the field of DIAs. The main goal pursued in this work was the optimal capacity allocation and management of big data deployments with performance guarantees, both at design and run time, yet a lot of attention was dedicated to other aspects that play an enabling role.

In particular, Chapter 3 focuses on the analysis of various performance models obtained via a series of formalisms, such as QNs, SWNs, and ML. Exploring such a wide range of alternatives was made necessary by the varying requirements of the different optimization procedures, specifically in terms of execution times of the searches themselves, as well as accuracy. To begin with, our optimization procedures are initially formulated through mathematical programming. Algebraic models can be easily added as constraints to the formulation, thus enabling the use of KKT conditions, which provide in closed form a sensible starting point for the search. In this case ML methods are perfectly fit for the requirement, as their output is an algebraic formula involving application parameters and system configurations. However, their big disadvantage is a lack of reliability when applied outside of the range covered by the data available at the time of training: since it is not possible to guarantee that the optimal configuration belongs to that domain, even more so at design time, we also adopt simulation-based techniques to complement ML. During the simulation-optimization approach, QNs, SWNs, or dagSim play their role, thanks to both their insensitivity to the evaluation range and accuracy, with relative errors settling within 3 % and 10 % on average, as reported in Chapter 5. Such results positively answer to research question 1.

Alongside their exploitation to support optimization, we also discuss the use of performance models to assess design choices, in line with the topic of research question 4. For example, Section 3.3.2 extends a basic performance model for Hadoop by considering the effects of spot instances failures caused by cloud providers' decisions to reclaim data center capacity under increased load. Chapter 5 also reports the results of some analyses based on this SWN,

with the goal of understanding the possible cost savings and the corresponding performance degradation enabled by this peculiar pricing model.

A further contribution of Chapter 3 is a pair of approaches to performance modeling of CNN applications deployed on GPGPUs. In this scenario, due to the extreme level of parallelism that rules out most simulation-based formalisms, we opt for two alternative ML models. The first one, the so called per layer model, is based on the derivation of the computational complexity of each type of layer used in CNNs, so as to apply linear regression and link complexity to layer execution times. In this way we obtain a set of models general enough to be applied to new CNNs that are not part of the training set. As a second approach, we also motivate the adoption of an end to end model based on two fundamental parameters, batch size and number of iterations, in order to predict performance when an application is already deployed on a specific system. This method enables a higher accuracy, but at the expense of decreased generality, since the obtained model is hardwired to a particular pair CNN-GPU. In both cases, the observed accuracy is good and, along the lines of research question 1, these performance models may be adopted in future work at the base of optimization procedures.

After presenting performance models, in Chapter 4 we shifted to the optimization methods topic, following research question 2. Thanks to the collaboration with two research projects, DICE H2020 and EUBra-BIGSEA, we investigated both design and run time problems. At first, we focused on design time issues, such as capacity allocation in cloud environments, with a design space exploration that considers also the choice among different alternative VM types to support the workload. After performing the basic choice on the instance type via ML models, the optimization proceeds with a search technique relying on third party simulators and terminates by returning the minimum cost configuration able to satisfy QoS constraints. Later on, we turned our attention to run time problems, like minimizing the overall tardiness of a set of jobs when the current workload exceeds what had been forecast, for instance due to an unforeseen peak of requests. In this scenario there are stringent constraints on the execution time of the optimization procedure, hence we need to consider only the most performant simulators to support the search phase. For this reason, in this case we adopt dagSim or Lundstrom, fast simulators developed by research partners in the frame of EUBra-BIGSEA.

We evaluated these optimization methods in Chapter 6. D-SPACE4Cloud allows for several considerations at design time, for instance assessing the effect of QoS constraints or concurrency levels on the costs incurred to support big data applications. Properly deciding the VM type where to run a workload leads to cost savings up to 36.4 %, with the additional consideration that it is not possible to single out one dominant choice but, as hinted in research question 3, actual workloads, concurrency levels, and QoS constraints play a role in determining the optimal configuration. In order to highlight how D-SPACE4Cloud can be useful during DIA design, we reported the outcomes of a case study for NETF Big Blu, a tax fraud detection application, showing how our tool was used to investigate in early stages the possible impact on costs of different privacy preserving mechanisms: this case study is another example of application of the proposed techniques in designing DIAs, according to the focus of research question 4. Shifting focus to run time aspects, we also assessed the quality of the proposed solution for the minimum weighted

tardiness problem. It appears that OPT_JR can sensibly modulate the solution based on system pressure, with the ability to return it within minutes also under a heavy load. Further, such solutions show a 21 % relative error with respect to measurements on a real system.

In the future, the research carried out so far will be extended in various directions. One of these will be the validation of our performance models against ML and DL workloads, an appealing scenario given their widespread adoption in the industry for a range of applications. Furthermore, via extending the underlying performance predictors, also the optimization methods will be made compatible with such new workloads. Another interesting research branch to explore is the optimal sizing of GPU-based systems, possibly not only for CNNs, but also for other applications. Similarly, it will be relevant to extend the investigation beyond one-GPU nodes, taking into account both multi-GPU machines and clusters of such GPU-enabled installations.

Acronyms

AES Advanced Encryption Standard. 123, 124, 126

AM analytical model. vi, 6, 22, 25, 27, 28, 30, 31, 52–56, 94, 95, 97, 98, 100, 102

API application programming interface. 4, 13, 20, 21, 74

CDF cumulative distribution function. 89

CNN convolutional neural network. xiii, 2, 4, 5, 7, 22, 25, 26, 33, 34, 44–51, 83, 104–109, 140, 141

CPN colored Petri net. 23

CPU central processing unit. 2, 5, 6, 17, 18, 23, 25–27, 34, 63, 65, 66, 74, 84, 93, 94, 97, 105, 112, 130, 134

CV coefficient of variation. 41, 43

DAG directed acyclic graph. xi, 4, 6, 8, 21, 29, 30, 34, 36, 43, 54, 65, 74, 91–93, 102, 103

DB database. 3, 20, 111, 116, 122, 123, 130

DDSM DICE Platform, Technology, and Deployment Specific Model. xi, 60, 62, 63

DES discrete event simulator. 7, 62, 74, 91

DIA data intensive application. v, 1–10, 13, 22, 30, 36, 60–63, 65, 71, 73, 81, 91, 99, 136, 139, 140

DL deep learning. 4–7, 18, 25, 31, 44, 141

DPIM DICE Platform Independent Model. 60

DT decision tree. 28

DTSM DICE Platform and Technology Specific Model. 60, 62

FCR finite capacity region. 35, 36

FIFO first in, first out. 5, 7, 18, 34–36, 38

FSPN fluid stochastic Petri net. xi, 39, 40

GPGPU general purpose graphics processing unit. 5–7, 26, 44, 45, 48–51, 109, 140

GPS Global Positioning System. 128

GPU graphics processing unit. xiii, 18, 25, 26, 29, 45, 48, 50, 83, 105, 106, 109, 140, 141

HDFS Hadoop Distributed File System. xi, 14–17, 21, 23, 84

HQL Hive Query Language. 20, 23

I/O input/output. 3, 19, 25

IaaS infrastructure as a service. 22, 63, 64, 68, 76

ICT information and communication technology. 22

IDE integrated development environment. 60, 61, 81

IT information technology. 2

JSON JavaScript object notation. 20

KB knowledge base. 28, 52, 53, 55, 56, 95, 100

KKT Karush-Kuhn-Tucker. 56, 69, 77, 79, 81, 139

MAPE mean average percentage error. 28, 53, 55, 95, 98, 102, 103, 105–109

MC Markov chain. xi, 23, 24, 30, 34, 39, 42

MINLP mixed integer nonlinear programming. 62, 68

ML machine learning. v, vi, 6–8, 13, 18–20, 25–28, 30, 31, 33, 34, 52–57, 62, 65, 66, 68, 69, 76, 77, 79, 81, 93–95, 97–104, 108, 109, 116, 119, 128, 139–141

MVP minimum viable product. 121, 122, 136

NN neural network. 4, 26, 27, 29

OS operating system. 14, 15, 24

P8 POWER8. 7, 112, 121, 130, 131

PBS Portable Batch System. 84

PDF probability distribution function. 41, 86, 100

PN Petri net. 23–25, 30

POSIX Portable Operating System Interface. 15

QN queueing network. v, vi, xi, xiii, 7, 8, 23–25, 27, 30, 33–36, 54, 56, 57, 62, 83, 85–87, 100, 101, 108, 139

QoS quality of service. v, vi, 1, 2, 4, 8, 9, 27, 28, 60–62, 64, 73–76, 116, 117, 120, 130, 137, 140

RAM random access memory. 84, 106, 111, 112, 123, 128, 130

RDD resilient distributed dataset. 4, 6, 10, 20, 21

RL reinforcement learning. 27

SLA service level agreement. 8, 10, 23, 26, 29–31, 76, 81, 109, 119

SPN stochastic Petri net. 8, 23, 39, 62

SQL Structured Query Language. 19, 20, 24, 91

SSD solid state drive. 3, 130

SVM support vector machine. 51

SVR support vector regression. 26, 54, 55, 62, 69, 94

SWN stochastic well formed net. v, vi, xi, xiii, 7, 33–38, 56, 57, 83, 85–88, 108, 109, 139

TAS Transactional Auto Scaler. 27

UML Unified Modeling Language. 60, 62

vCPU virtual central processing unit. 80, 83, 111, 123, 130

VM virtual machine. vi, 7, 8, 10, 36–39, 52, 61–72, 74–77, 79–81, 83, 87, 88, 92, 111, 112, 114, 116–118, 123, 128, 130, 135, 136, 140

YARN Yet Another Resource Negotiator. 6, 8, 17, 18, 20, 28, 29, 35, 39, 41, 53, 63, 65, 73, 88

Bibliography

[1] Longendri Aguilera-Mendoza and Monica T. Llorente-Quesada. "Modeling and Simulation of Hadoop Distributed File System in a Cluster of Workstations." English. In: *Model and Data Engineering*. Vol. 8216. 2013, pp. 1–12. ISBN: 978-3-642-41365-0.

[2] Syed Thouheed Ahmed and Dmitri Loguinov. "On the Performance of MapReduce: A Stochastic Approach." In: *IEEE International Conference on Big Data*. IEEE. 2014, pp. 49–54.

[3] A. Brito et al. *D3.4 EUBra-BIGSEA QoS Infrastructure Services Intermediate Version*. 2017. URL: http://www.eubra-bigsea.eu/sites/default/files/D3.4%20EUBra-BIGSEA%20QoS%20infrastructure%20services.pdf.

[4] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. "Software Architecture Optimization Methods: A Systematic Literature Review." In: *Software Engineering, IEEE Transactions on* PP.99 (2013), pp. 1–1.

[5] Hanieh Alipour, Yan Liu, Abdelwahab Hamou-Lhadj, and Ian Gorton. "Model Driven Performance Simulation of Cloud Provisioned Hadoop MapReduce Applications." In: *Proceedings of the 8th International Workshop on Modeling in Software Engineering*. MiSE '16. Austin, Texas: ACM, 2016, pp. 48–54. ISBN: 978-1-4503-4164-6. DOI: 10.1145/2896982.2896989.

[6] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics." In: *NSDI*. 2017.

[7] Jussara Almeida, Danilo Ardagna, Igor Ataíde, Enrico Barbierato, Ignacio Blanquer, Sergio López, Túlio Braga, Andrey Brito, Ana Paula Couto, Athanasia Evangelinou, Iury Ferreira, Armstrong Goes, Marco Gribaudo, Raffaela Mirandola, and Fábio Morais. *D3.5 EUBra-BIGSEA QoS Infrastructure Services Final Version*. Oct. 2017. URL: http://eubra-bigsea.eu/sites/default/files/D3.5%20EUBra-BIGSEA%20QoS%20infrastructure%20services%20final%20version.pdf.

[8] Danilo Ardagna, Enrico Barbierato, Athanasia Evangelinou, Eugenio Gianniti, Marco Gribaudo, Túlio B. M. Pinto, Anna Guimarães, Ana Paula Couto da Silva, and Jussara M. Almeida. "Performance Prediction of Cloud-Based Big Data Applications." In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE

'18. Berlin, Germany: ACM, 2018, pp. 192–199. ISBN: 978-1-4503-5095-2. DOI: 10.1145/3184407.3184420.

[9] Danilo Ardagna, Simona Bernardi, Eugenio Gianniti, Soroush Karimian Aliabadi, Diego Perez-Palacin, and José Ignacio Requeno. "Modeling Performance of Hadoop Applications: A Journey from Queueing Networks to Stochastic Well Formed Nets." In: *16th International Conference on Algorithms and Architectures for Parallel Processing*. Ed. by Jesús Carretero, Javier García Blas, Ryan K. L. Ko, Peter Mueller, and Koji Nakano. Vol. 10048. Lecture Notes in Computer Science. Springer, Dec. 2016, pp. 599–613. ISBN: 978-3-319-49582-8. DOI: 10.1007/978-3-319-49583-5_47.

[10] Danilo Ardagna, Michele Ciavotta, and Mauro Passacantando. "Generalized Nash Equilibria for the Service Provisioning Problem in Multi-Cloud Systems." In: *IEEE Trans. Services Computing* 10.3 (2017), pp. 381–395.

[11] Danilo Ardagna, Carlo Ghezzi, and Raffaela Mirandola. "Rethinking the Use of Models in Software Architecture." In: *QoSA*. 2008.

[12] Danilo Ardagna, Eugenio Gianniti, Jacopo Rigoli, Safia Kalwar, and Giuliano Casale. *DICE Optimization Tools — Final Version*. July 2017. URL: http://wp.doc.ic.ac.uk/dice-h2020/wp-content/uploads/sites/75/2017/08/D3.9_DICE-optimization-tools-Final-version.pdf.

[13] Sylvain Arlot and Alain Celisse. "A Survey of Cross-Validation Procedures for Model Selection." In: *Statistics Surveys*. 2010, pp. 40–79.

[14] Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. "Model-Driven Continuous Deployment for Quality DevOps." In: *QUDOS*. 2016.

[15] Ehsan Ataie, Eugenio Gianniti, Danilo Ardagna, and Ali Movaghar. "A Combined Analytical Modeling Machine Learning Approach for Performance Prediction of MapReduce Jobs in Cloud Environment." In: *SYNASC* (Timisoara, Romania). 2016.

[16] Soheib Baarir, Marco Beccuti, Davide Cerotti, Massimiliano De Pierro, Susanna Donatelli, and Giuliana Franceschinis. "The GreatSPN Tool: Recent Enhancements." In: *ACM SIGMETRICS PER* 36.4 (2009), pp. 4–9.

[17] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *CoRR* abs/1409.0473 (2014).

[18] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. "Comparative Study of Caffe, Neon, Theano, and Torch for Deep Learning." In: *CoRR* abs/1511.06435 (2015).

[19] Simonetta Balsamo, Peter G. Harrison, and Andrea Marin. "Methodological Construction of Product-Form Stochastic Petri Nets for Performance Evaluation." In: *Journal of Systems and Software* 85.7 (2012), pp. 1520–1539. DOI: 10.1016/j.jss.2011.11.1042.

[20] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. "Modeling Apache Hive Based Applications in Big Data Architectures." In: *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*. ValueTools '13. Torino, Italy: ICST, 2013, pp. 30–38. ISBN: 978-1-936968-48-0. DOI: 10.4108/icst.valuetools.2013.254398.

[21] Enrico Barbierato, Marco Gribaudo, and Mauro Iacono. "Modeling Hybrid Systems in SIMTHESys." In: *Proceedings of the 8th International Workshop on Practical Applications of Stochastic Modelling*. Münster, Germany, 2016.

[22] Shouvik Bardhan and Daniel A. Menascé. "Queuing Network Models to Predict the Completion Time of the Map Phase of MapReduce Jobs." In: *Proc. International Computer Measurement Group Conf.* (Las Vegas, NV). Dec. 2012.

[23] S. Becker, H. Koziolek, and R. Reussner. "The Palladio Component Model for Model-driven Performance Prediction." In: *Journal of Systems and Software* 82.1 (2009), pp. 3–22.

[24] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. "JMT: Performance Engineering Tools for System Modeling." In: *SIGMETRICS Perform. Eval. Rev.* 36.4 (2009), pp. 10–15. ISSN: 0163-5999. DOI: 10.1145/1530873.1530877.

[25] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev. "Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-Based Architectures." In: *Software Engineering, IEEE Transactions on* 41.2 (Feb. 2015), pp. 157–175. ISSN: 0098-5589. DOI: 10.1109/TSE.2014.2362755.

[26] Dario Bruneo, Francesco Longo, Rahul Ghosh, Marco Scarpa, Antonio Puliafito, and Kishor S. Trivedi. "Analytical Modeling of Reactive Autonomic Management Techniques in IaaS Clouds." In: *IEEE CLOUD*. 2015.

[27] Giuliano Casale, Danilo Ardagna, Matej Artac, Franck Barbier, Elisabetta Di Nitto, Alexis Henry, Gabriel Iuhasz, Christophe Joubert, José Merseguer, Victor Ion Munteanu, Juan Fernando Pérez, Dana Petcu, Matteo Rossi, Craig Sheridan, Ilias Spais, and Daniel Vladuic. "DICE: Quality-Driven Development of Data-Intensive Cloud Applications." In: *International Workshop on Modeling in Software Engineering* (Florence, Italy). Ed. by Jeff Gray, Marsha Chechik, Vinay Kulkarni, and Richard F. Paige. IEEE Computer Society, May 2015, pp. 78–83. ISBN: 978-1-4673-7055-4. DOI: 10.1109/MiSE.2015.21. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7166141.

[28] Aniello Castiglione, Marco Gribaudo, Mauro Iacono, and Francesco Palmieri. "Exploiting Mean Field Analysis to Model Performances of Big Data Architectures." In: *Future Generation Computer Systems* 37 (2014), pp. 203–211. ISSN: 0167-739X. DOI: 10.1016/j.future.2013.07.016.

[29] C.-C. Chang and C.-J. Lin. "LIBSVM: A Library for Support Vector Machines." In: *ACM Transactions on Intelligent Systems and Technology* 2.27 (2011), pp. 1–27.

[30] Tatsuhiro Chiba and Tamiya Onodera. "Workload Characterization and Optimization of TPC-H Queries on Apache Spark." In: *ISPASS*. IEEE, Apr. 2016. DOI: 10.1109/ISPASS.2016.7482079.

[31] Wesley W. Chu, Chi-Man Sit, and Kin K. Leung. "Task Response Time for Real-Time Distributed Systems with Resource Contentions." In: *IEEE Trans. Softw. Eng.* 17.10 (), pp. 1076–1092. ISSN: 0098-5589. DOI: 10.1109/32.99195.

[32] Michele Ciavotta, Eugenio Gianniti, and Danilo Ardagna. "Capacity Allocation for Big Data Applications in the Cloud." In: *ICPE'17 Companion* (L'Aquila, Italy). Apr. 2017, pp. 175–176.

[33] Michele Ciavotta, Eugenio Gianniti, and Danilo Ardagna. "D-SPACE4Cloud: A Design Tool for Big Data Applications." In: *ICA3PP* (Granada, Spain). Lecture Notes in Computer Science 10048. Springer, Dec. 2016, pp. 614–629.

[34] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. "Reservation-Based Scheduling: If You're Late Don't Blame Us!" In: *Proceedings of the ACM Symposium on Cloud Computing*. SOCC '14. Seattle, WA, USA: ACM, 2014. ISBN: 978-1-4503-3252-1. DOI: 10.1145/2670979.2670981.

[35] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. "BOAT: Building Auto-Tuners with Structured Bayesian Optimization." In: *WWW*. 2017.

[36] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Commun. ACM* 51.1 (2008), pp. 107–113.

[37] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *6th Symposium on Operating Systems Design and Implementation*. 2004, pp. 137–149.

[38] Christina Delimitrou and Christos Kozyrakis. "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters." In: *SIGPLAN Not.* 48.4 (Mar. 2013), pp. 77–88. ISSN: 0362-1340.

[39] Christina Delimitrou and Christos Kozyrakis. "Quasar: Resource-Efficient and QoS-Aware Cluster Management." In: *ACM SIGPLAN Notices*. Vol. 49. 4. ACM. 2014, pp. 127–144.

[40] Christina Delimitrou and Christos Kozyrakis. "Quasar: Resource-efficient and QoS-aware Cluster Management." In: *ASPLOS*. 2014.

[41] H. Derrick. *Survey Shows Huge Popularity Spike for Apache Spark*. 2015. URL: http://fortune.com/2015/09/25/apache-spark-survey.

[42] P Di Sanzo, F Quaglia, B Ciciani, A Pellegrini, D Didona, P Romano, R Palmieri, and S Peluso. "A Flexible Framework for Accurate Simulation of Cloud In-Memory Data Stores." In: *Simulation Modelling Practice and Theory* 58 (2015), pp. 219–238.

[43] Diego Didona, Pascal Felber, Derin Harmanci, Paolo Romano, and Jörg Schenker. "Identifying the Optimal Level of Parallelism in Transactional Memory Applications." In: *Computing* 97.9 (2015), pp. 939–959.

[44] Diego Didona, Francesco Quaglia, Paolo Romano, and Ennio Torre. "Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning." In: *6th ACM/SPEC International Conference on Performance Engineering*. 2015, pp. 145–156.

[45] Diego Didona and Paolo Romano. "Hybrid Machine Learning/Analytical Models for Performance Prediction: A Tutorial." In: *International Conference on Performance Engineering*. ACM/SPEC. 2015, pp. 341–344.

[46] Diego Didona and Paolo Romano. "On Bootstrapping Machine Learning Performance Predictors via Analytical Models." In: (2014). arXiv: 1410.5102v1 [cs.PF].

[47] Diego Didona, Paolo Romano, Sebastiano Peluso, and Francesco Quaglia. "Transactional Auto Scaler: Elastic Scaling of Replicated In-Memory Transactional Data Grids." In: *ACM Transactions on Autonomous and Adaptive Systems* 9.2 (2014), pp. 1–32.

[48] Daniel J. Dubois and Giuliano Casale. "OptiSpot: Minimizing Application Deployment Cost Using Spot Cloud Resources." In: *Cluster Computing* (2016), pp. 1–17.

[49] John Ganz and David Reinsel. *The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East — United States*. Tech. rep. IDC, Feb. 2013. URL: https://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf (visited on 04/07/2015).

[50] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[51] Eugenio Gianniti, Danilo Ardagna, Michele Ciavotta, and Mauro Passacantando. "A Game-Theoretic Approach for Runtime Capacity Allocation in MapReduce." In: *CCGrid* (Madrid, Spain). May 2017, pp. 1080–1089.

[52] Eugenio Gianniti, Alessandro Maria Rizzi, Enrico Barbierato, Marco Gribaudo, and Danilo Ardagna. "Fluid Petri Nets for the Performance Evaluation of MapReduce and Spark Applications." In: *ACM Performance Evaluation Review* 44.4 (Mar. 2017), pp. 23–36. DOI: 10.1145/3092819.3092824.

[53] Eugenio Gianniti, Alessandro Maria Rizzi, Enrico Barbierato, Marco Gribaudo, and Danilo Ardagna. "Fluid Petri Nets for the Performance Evaluation of MapReduce Applications." In: *InfQ* (Taormina, Italy). Oct. 2016.

[54] G. P. Gibilisco, M. Li, L. Zhang, and D. Ardagna. "Stage Aware Performance Modeling of DAG Based in Memory Analytic Platforms." In: *Cloud*. 2016.

[55] M. A. Greene and K. Sreekanti. *Big Data in the Enterprise: We Need an "Easy Button" for Hadoop*. 2016. URL: http://www.oreilly.com/pub/e/3643.

[56] Marco Gribaudo and Miklós Telek. "Fluid Models in Performance Analysis." In: *International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. Ed. by Marco Bernardo and Jane Hillston. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 271–317. ISBN: 978-3-540-72522-0. DOI: 10.1007/978-3-540-72522-0_7.

[57] Lei Gu and Huan Li. "Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark." In: *HPCC-EUC*. IEEE, Nov. 2013. DOI: 10.1109/HPCC.and.EUC.2013.106.

[58] Stefan Hadjis, Ce Zhang, Ioannis Mitliagkas, and Christopher Ré. "Omnivore: An Optimizer for Multi-Device Deep Learning on CPUs and GPUs." In: *CoRR* abs/1606.04487 (2016).

[59] Herodotos Herodotou, Fei Dong, and Shivnath Babu. "No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics." In: *SOCC*. Cascais, Portugal, 2011. DOI: 10.1145/2038916.2038934.

[60] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. "Starfish: A Self-Tuning System for Big Data Analytics." In: *CIDR*. 2011.

[61] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center." In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 295–308. URL: http://dl.acm.org/citation.cfm?id=1972457.1972488.

[62] Engin Ipek, Sally A. McKee, Bronis R. de Supinski, and Rich Caruana. "Efficiently Exploring Architectural Design Spaces via Predictive Modeling." In: *12th International Conference on Architectural Support for Programming Languages and Operating Systems*. 2006, pp. 195–206.

[63] Florin Isaila, Prasanna Balaprakash, Stefan M Wild, Dries Kimpe, Rob Latham, Rob Ross, and Paul Hovland. "Collective I/O Tuning Using Analytical and Machine Learning Models." In: *International Conference on Cluster Computing*. IEEE. 2015, pp. 128–137.

[64] Muhammed Tawfiqul Islam, Shanika Karunasekera, and Rajkumar Buyya. "dSpark: Deadline-Based Resource Allocation for Big Data Applications in Apache Spark." In: *13th IEEE International Conference on e-Science*. IEEE Computer Society, 2017, pp. 89–98. ISBN: 978-1-5386-2686-3. DOI: 10.1109/eScience.2017.21.

[65] H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. "Big Data and Its Technical Challenges." In: *Commun. ACM* 57.7 (July 2014), pp. 86–94. ISSN: 0001-0782.

[66] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems." In: *International Journal on Software Tools for Technology Transfer* 9.3-4 (2007), pp. 213–254.

[67] Wenhao Jia, Kelly A. Shaw, and Margaret Martonosi. "Stargazer: Automated Regression-based GPU Design Space Exploration." In: *ISPASS*. IEEE. Apr. 2012. DOI: 10.1109/ISPASS.2012.6189201.

[68] Hui Jin, Kan Qiao, Xian-He Sun, and Ying Li. "Performance under Failures of MapReduce Applications." In: *CCGrid*. 2011.

[69] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. "Morpheus: Towards Automated SLOs for Enterprise Clusters." In: *12th USENIX Symposium on Operating Systems Design and Implementation*. OSDI 16. Savannah, GA: USENIX Association, 2016, pp. 117–134. ISBN: 978-1-931971-33-1. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/jyothi.

[70] Safia Kalwar, Eugenio Gianniti, Joas Yannick Kinouani, Youssef Ridene, and Danilo Ardagna. "Performance Degradation and Cost Impact Evaluation of Privacy Preserving Mechanisms in Big Data Systems." In: *New Frontiers in Quantitative Methods in Informatics*. Ed. by Simonetta Balsamo, Andrea Marin, and Enrico Vicario. Vol. 825. Communications in Computer and Information Science. Springer, 2017, pp. 82–96. ISBN: 978-3-319-91631-6. DOI: 10.1007/978-3-319-91632-3_7.

[71] Karthik Kambatla, Giorgos Kollias, Vipin Kumar, and Ananth Grama. "Trends in Big Data Analytics." In: *Journal of Parallel and Distributed Computing* 74.7 (2014), pp. 2561–2573. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2014.01.003.

[72] Andrew Kerr, Gregory Diamos, and Sudhakar Yalamanchili. "Modeling GPU-CPU Workloads and Systems." In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. GPGPU-3. Pittsburgh, Pennsylvania, USA: ACM, 2010, pp. 31–42. ISBN: 978-1-60558-935-0. DOI: 10.1145/1735688.1735696.

[73] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. "PerOpteryx: Automated Application of Tactics in Multi-objective Software Architecture Optimization." In: *QoSA*. QoSA-ISARCS '11. Boulder, Colorado, USA: ACM, 2011, pp. 33–42. ISBN: 978-1-4503-0724-6. DOI: 10.1145/2000259.2000267.

[74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *NIPS*. Vol. 1. Dec. 2012, pp. 1097–1105.

[75] Johannes Kross and Helmut Krcmar. "Model-Based Performance Evaluation of Batch and Stream Applications for Big Data." In: *MASCOTS*. 2017.

[76] Palden Lama and Xiaobo Zhou. "AROMA: Automated Resource Allocation and Configuration of MapReduce Environment in the Cloud." In: *9th International Conference on Autonomic Computing*. 2012, pp. 63–72.

[77] Douglas Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. Tech. rep. META Group, 2012.

[78] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative System Performance. Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984. URL: http://homes.cs.washington.edu/~lazowska/qsp/ (visited on 04/07/2015).

[79] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. "Parallel Data Processing with MapReduce: A Survey." In: *SIGMOD Rec.* 40.4 (Jan. 2012), pp. 11–20. ISSN: 0163-5808. DOI: 10.1145/2094114.2094118.

[80] Min Li, Jian Tan, Yandong Wang, Li Zhang, and Valentina Salapura. "SparkBench: A Comprehensive Benchmarking Suite for in Memory Data Analytic Platform Spark." In: *Proceedings of the 12th ACM International Conference on Computing Frontiers*. CF '15. Ischia, Italy: ACM, 2015, 53:1–53:8. ISBN: 978-1-4503-3358-0. DOI: 10.1145/2742854.2747283.

[81] De-Ron Liang and Satish K. Tripathi. "On Performance Prediction of Parallel Computations with Precedent Constraints." In: *IEEE Transactions on Parallel and Distributed Systems* 11.5 (May 2000), pp. 491–508. ISSN: 1045-9219. DOI: 10.1109/71.852402.

[82] Min Lin, Qiang Chen, and Shuicheng Yan. "Network in Network." In: *CoRR* abs/1312.4400 (2013).

[83] Minghong Lin, Li Zhang, Adam Wierman, and Jian Tan. "Joint Optimization of Overlapping Phases in MapReduce." In: *SIGMETRICS Performance Evaluation Review* 41.3 (2013), pp. 16–18. DOI: 10.1145/2567529.2567534.

[84] Xuelian Lin, Zide Meng, Chuan Xu, and Meng Wang. "A Practical Performance Model for Hadoop MapReduce." In: *CLUSTER*. IEEE. 2012, pp. 231–239.

[85] Xuelian Lin, Zide Meng, Chuan Xu, and Meng Wang. "A Practical Performance Model for Hadoop MapReduce." In: *International Conference on Cluster Computing Workshops*. IEEE. 2012. DOI: 10.1109/ClusterW.2012.24.

[86] Weiguo Liu, Wolfgang Muller-Wittig, and Bertil Schmidt. "Performance Predictions for General-Purpose Computation on GPUs." In: *ICPP*. IEEE. Sept. 2007. DOI: 10.1109/ICPP.2007.67.

[87] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. "Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping." In: *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 42. New York, New York: ACM, 2009, pp. 45–55. ISBN: 978-1-60558-798-1. DOI: 10.1145/1669112.1669121.

[88] V. W. Mak and S. F. Lundstrom. "Predicting Performance of Parallel Computations." In: *IEEE Trans. Parallel Distrib. Syst.* 1.3 (July 1990), pp. 257–270. ISSN: 1045-9219. DOI: 10.1109/71.80155.

[89] Marzieh Malekimajd, Danilo Ardagna, Michele Ciavotta, Eugenio Gianniti, Mauro Passacantando, and Alessandro Maria Rizzi. "An Optimization Framework for the Capacity Allocation and Admission Control of MapReduce Jobs in Cloud Systems." In: *The Journal of Supercomputing* (May 2018). ISSN: 1573-0484. DOI: 10.1007/s11227-018-2426-2.

[90] Marzieh Malekimajd, Danilo Ardagna, Michele Ciavotta, Alessandro Maria Rizzi, and Mauro Passacantando. "Optimal Map Reduce Job Capacity Allocation in Cloud Systems." In: *SIGMETRICS Perform. Eval. Rev.* 42.4 (June 2015), pp. 51–61. ISSN: 0163-5999. DOI: 10.1145/2788402.2788410.

[91] João Eugenio Marynowski, Altair Olivo Santin, and Andrey Ricardo Pimentel. "Method for Testing the Fault Tolerance of MapReduce Frameworks." In: *Computer Networks* 86 (2015), pp. 1–13.

[92] Ilias Mavridis and Helen Karatza. "Performance Evaluation of Cloud-Based Log File Analysis with Apache Hadoop and Apache Spark." In: *Journal of Systems and Software* 125 (2017), pp. 133–151. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.11.037.

[93] Rizwan Mian, Patrick Martin, and Jose Luis Vazquez-Poletti. "Provisioning Data Analytic Workloads in a Cloud." In: *Future Gener. Comput. Syst.* 29.6 (Aug. 2013), pp. 1452–1458. ISSN: 0167-739X.

[94] K. Morton, A Friesen, M. Balazinska, and D. Grossman. "Estimating the Progress of MapReduce Pipelines." In: *ICDE*. 2010.

[95] Kristi Morton, Magdalena Balazinska, and Dan Grossman. "ParaTimer: A Progress Indicator for MapReduce DAGs." In: *SIGMOD*. 2010. DOI: 10.1145/1807167.1807223.

[96] Randolf D. Nelson and Asser N. Tantawi. "Approximate Analysis of Fork/Join Synchronization in Parallel Queues." In: *IEEE Trans. Computers* 37.6 (1988), pp. 739–743. DOI: 10.1109/12.2213.

[97] OMG. *PEPA: Performance Evaluation Process Algebra*. 2015. URL: http://www.dcs.ed.ac.uk/pepa/tools/.

[98] Rasha Osman and Peter G. Harrison. "Approximating Closed Fork-Join Queueing Networks Using Product-Form Stochastic Petri-Nets." In: *J. Syst. Softw.* 110.C (Dec. 2015), pp. 264–278. ISSN: 0164-1212. DOI: 10.1016/j.jss.2015.08.036.

[99] Kay Ousterhout, Ryan Rasti, Sylvia Ratnasamy, Scott Shenker, and Byung Gon Chun. "Making Sense of Performance in Data Analytics Frameworks." In: *NSDI* (2015).

[100] Xinghao Pan, Shivaram Venkataraman, Zizheng Tai, and Joseph Gonzalez. "Hemingway: Modeling Distributed Optimization Algorithms." In: *CoRR* abs/1702.05865 (2017).

[101] Linh T. X. Phan, Zhuoyao Zhang, Qi Zheng, Boon Thau Loo, and Insup Lee. "An Empirical Analysis of Scheduling Techniques for Real-Time Cloud-based Data Processing." In: *SOCA*. 2011. ISBN: 978-1-4673-0318-7. DOI: 10.1109/SOCA.2011.6166240.

[102] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. 3rd. Springer Publishing Company, Incorporated, 2008. ISBN: 978-1-4614-2361-4.

[103] J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, and I Whalley. "Performance-driven Task Co-scheduling for MapReduce Environments." In: *NOMS*. 2010.

[104] Jorda Polo, Yolanda Becerra, David Carrera, Malgorzata Steinder, Ian Whalley, Jordi Torres, and Eduard Ayguadé. "Deadline-Based MapReduce Workload Management." In: *IEEE Trans. Network and Service Management* 10.2 (2013), pp. 231–244. DOI: 10.1109/TNSM.2012.122112.110163.

[105] B. Thirumala Rao and L. S. S. Reddy. "Survey on Improved Scheduling in Hadoop MapReduce in Cloud Environments." In: *CoRR* abs/1207.0780 (2012).

[106] Alessandro Maria Rizzi. "Support Vector Regression Model for Big-Data Systems." In: (Dec. 2016). arXiv: 1612.01458 [cs.DC].

[107] Diego Rughetti, Pierangelo Di Sanzo, Bruno Ciciani, Francesco Quaglia, and Sapienza Universit. "Analytical/ML Mixed Approach for Concurrency Regulation in Software Transactional Memory." In: *Cluster, Cloud and Grid Computing*. CCGrid. 2014, pp. 81–91.

[108] M Carmen Ruiz, Javier Calleja, and Diego Cazorla. "Petri Nets Formalization of Map/Reduce Paradigm to Optimise the Performance-Cost Tradeoff." In: *Trustcom/BigDataSE/ISPA*. Vol. 3. IEEE. 2015, pp. 92–99.

[109] Bikas Saha, Hitesh Shah, Siddharth Seth, Gopal Vijayaraghavan, Arun Murthy, and Carlo Curino. "Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications." In: *International Conference on Management of Data*. SIGMOD '15. ACM. Melbourne, Victoria, Australia, 2015, pp. 1357–1369. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742790.

[110] Tara N. Sainath, Brian Kingsbury, George Saon, Hagen Soltau, Abdelrahman Mohamed, George E. Dahl, and Bhuvana Ramabhadran. "Deep Convolutional Neural Networks for Large-scale Speech Tasks." In: *Neural Networks* 64 (2015), pp. 39–48.

[111] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. "Omega: Flexible, Scalable Schedulers for Large Compute Clusters." In: *SIGOPS European Conference on Computer Systems*. EuroSys. Prague, Czech Republic, 2013, pp. 351–364. URL: http://eurosys2013.tudos.org/wp-content/uploads/2013/paper/Schwarzkopf.pdf.

[112] Carter Shanklin. *Benchmarking Apache Hive 13 for Enterprise Hadoop*. June 2, 2014. URL: https://hortonworks.com/blog/benchmarking-apache-hive-13-enterprise-hadoop/.

[113] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *ICLR* (San Diego, CA, USA). May 2015. arXiv: 1409.1556v6 [cs.CV].

[114] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[115] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going Deeper with Convolutions." In: *CVPR* (Boston, MA, USA). IEEE, June 2015. DOI: 10.1109/CVPR.2015.7298594.

[116] Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. "On the Use of Hybrid Reinforcement Learning for Autonomic Resource Allocation." In: *Cluster Computing* 10 (2007), pp. 287–299.

[117] Eno Thereska and Gregory R Ganger. "IRONModel: Robust Performance Models in the Wild." In: *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 2008, pp. 253–264.

[118] Fengguang Tian and Keke Chen. "Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds." In: *CLOUD*. 2011.

[119] Mirco Tribastone, Stephen Gilmore, and Jane Hillston. "Scalable Differential Analysis of Process Algebra Models." In: *IEEE Transactions on Software Engineering* 38.1 (2012), pp. 205–219. ISSN: 0098-5589. DOI: 10.1109/TSE.2010.82.

[120] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. "Apache Hadoop YARN: Yet Another Resource Negotiator." In: *SOCC*. 2013.

[121] Shivaram Venkataraman, Zongheng Yang, Michael J Franklin, Benjamin Recht, and Ion Stoica. "Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics." In: *NSDI*. 2016.

[122] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments." In: *Proceedings of the Eighth International Conference on Autonomic Computing*. June 2011.

[123] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. "Profiling and Evaluating Hardware Choices for MapReduce Environments: An Application-aware Approach." In: *Performance Evaluation* 79 (2014), pp. 328–344. DOI: 10.1016/j.peva.2014.07.020.

[124] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. "Large-Scale Cluster Management at Google with Borg." In: *EuroSys*. 2015.

[125] Emanuel Vianna, Giovanni Comarela, Tatiana Pontes, Jussara M. Almeida, Virgílio A. F. Almeida, Kevin Wilkinson, Harumi A. Kuno, and Umeshwar Dayal. "Analytical Performance Models for MapReduce Workloads." In: *International Journal of Parallel Programming* 41.4 (2013), pp. 495–525. DOI: 10.1007/s10766-012-0227-4.

[126] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion." In: *Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[127] Kewen Wang and Mohammad Maifi Hasan Khan. "Performance Prediction for Apache Spark Platform." In: *HPCC-CSS-ICESS*. IEEE, Aug. 2015. DOI: 10.1109/HPCC-CSS-ICESS.2015.246.

[128] Kewen Wang, Mohammad Maifi Hasan Khan, Nhan Nguyen, and Swapna Gokhale. "Modeling Interference for Apache Spark Jobs." In: *CLOUD*. IEEE, 2016. DOI: 10.1109/CLOUD.2016.0063.

[129] Weina Wang, Kai Zhu, Lei Ying, Jian Tan, and Li Zhang. "MapTask Scheduling in MapReduce With Data Locality: Throughput and Heavy-Traffic Optimality." In: *IEEE/ACM Transactions on Networking* 24.1 (2016), pp. 190–203.

[130] Yandong Wang, Li Zhang, Yufei Ren, and Wei Zhang. "Nexus: Bringing Efficient and Scalable Training to Deep Learning Frameworks." In: *MASCOTS*. Banff, Canada, Sept. 2017, pp. 12–21.

[131] *Worldwide Semiannual Big Data and Analytics Spending Guide*. Mar. 2017. URL: https://www.idc.com/getdoc.jsp?containerId=IDC_P33195.

[132] Xiaoyong Xu and Maolin Tang. "A New Approach to the Cloud-based Heterogeneous MapReduce Placement Problem." In: *IEEE Transactions on Services Computing* 9.6 (2015), pp. 862–871. DOI: 10.1109/TSC.2015.2433914.

[133] Feng Yan, Ludmila Cherkasova, Zhuoyao Zhang, and Evgenia Smirni. "Optimizing Power and Performance Trade-offs of MapReduce Job Processing with Heterogeneous Multi-Core Processors." In: *CLOUD*. 2014. DOI: 10.1109/CLOUD.2014.41.

[134] Xiao Yang and Jianling Sun. "An Analytical Performance Model of MapReduce." In: *CCIS*. IEEE. 2011.

[135] Nezih Yigitbasi, Theodore L. Willke, Guangdeng Liao, and Dick Epema. "Towards Machine Learning-Based Auto-tuning of MapReduce." In: *IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. 2013, pp. 11–20.

[136] Xiaolong Yu and Wei Li. "Performance Modelling and Analysis of MapReduce/Hadoop Workloads." In: *LANMAN*. 2015.

[137] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing." In: *9th USENIX Conference on Networked Systems Design and Implementation*. NSDI'12. San Jose, CA: USENIX Association, 2012, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=2228298.2228301.

[138] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. "Spark: Cluster Computing with Working Sets." In: *2nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, 2010, pp. 10–10. URL: http://dl.acm.org/citation.cfm?id=1863103.1863113.

[139] Matei Zaharia, Patrick Wendell, Andy Konwinski, and Holden Karau. *Learning Spark*. O'Reilly Media, Inc., 2015. ISBN: 978-1-4493-5862-4.

[140] Wei Zhang, Sundaresan Rajasekaran, Shaohua Duan, Timothy Wood, and Mingfa Zhu. "Minimizing Interference and Maximizing Progress for Hadoop Virtual Machines." In: *SIGMETRICS Performance Evaluation Review* 42.4 (2015), pp. 62–71. DOI: 10.1145/2788402.2788411.

[141] Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo. "Exploiting Cloud Heterogeneity to Optimize Performance and Cost of MapReduce Processing." In: *SIGMETRICS Performance Evaluation Review* 42.4 (2015), pp. 38–50. DOI: 10.1145/2788402.2788409.

[142] Zhuoyao Zhang, Ludmila Cherkasova, and Boon Thau Loo. "Exploiting Cloud Heterogeneity to Optimize Performance and Cost of MapReduce Processing." In: *SIGMETRICS Performance Evaluation Review* 42.4 (2015), pp. 38–50. DOI: 10.1145/2788402.2788409.

[143] Zhuoyao Zhang, Ludmila Cherkasova, Abhishek Verma, and Boon Thau Loo. "Automated Profiling and Resource Management of Pig Programs for Meeting Service Level Objectives." In: *ICAC*. San Jose, California, USA, 2012. ISBN: 978-1-4503-1520-3. DOI: 10.1145/2371536.2371546.