We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

5,000
Open access books available

125,000
International authors and editors

140M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter

Looking at Data Science through the Lens of Scheduling and Load Balancing

Diórgenes Eugênio da Silveira, Eduardo Souza dos Reis, Rodrigo Simon Bavaresco, Marcio Miguel Gomes, Cristiano André da Costa, Jorge Luis Victoria Barbosa, Rodolfo Stoffel Antunes, Alvaro Machado Júnior, Rodrigo Saad and Rodrigo da Rosa Righi

Abstract

The growth in data generated by private and public organizations leads to several opportunities to obtain valuable knowledge. In this scenario, data science becomes pertinent to define a structured methodology to extract valuable knowledge from raw data. It encompasses a heterogeneous group of techniques that challenge the implementation of a single platform capable of incorporating all the available resources. Thus, it is necessary to formulate a data science workflow based on different tools to extract knowledge from massive datasets. In this context, high-performance computing (HPC) provides the infrastructure required to optimize the processing time of data science workflows, which become a collection of tasks that must be efficiently scheduled to provide results in acceptable time intervals. While few studies explore the use of HPC for data science tasks, in the best of our knowledge, none conducts an in-depth analysis of scheduling and load balancing on such workflows. In this context, this chapter proposes an analysis of scheduling and load balancing from the perspective of data science scenarios. It presents concepts, environments, and tools to summarize the theoretical background required to define, assign, and execute data science workflows. Furthermore, we are also presenting new trends concerning the intersection of data science, scheduling, and load balance.

Keywords: scheduling, load balance, high-performance computing, data science, big data

1. Introduction

Private corporate networks, as well as the Internet, generate and share data at ever increasing rates. This unconstrained growth can easily lead disorganization and, as a consequence, missed opportunities to analyze and extract knowledge for these data. There is an essential difference between the concepts of data and information. Data cannot express something outside a particular field of expertise.

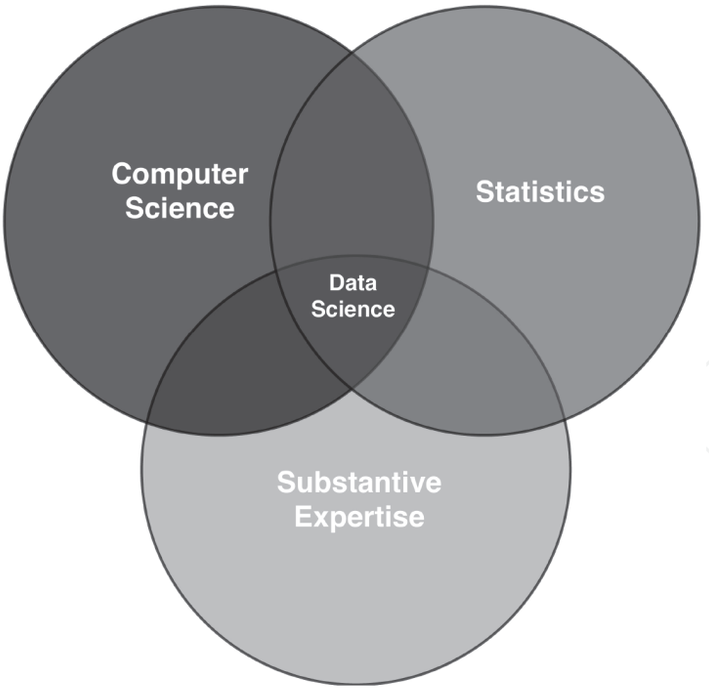

Figure 1.
Venn's diagram for correlating the influence of other research areas on data science.

In turn, information enables the coherent transmission of knowledge. Data science aims to close the gap between data and knowledge through the use of computational tools. More specifically, data science is a tool for converting raw data into knowledge [1]. The field of data science leverages many methods originating from computer science and statistics [2]. **Figure 1** illustrates a Venn's diagram that correlates the research areas with major influence in data science.

Although data science receives significant influence from expert knowledge, it is plausible to say that a data scientist knows more about computer science than a statistician and more about statistics than a computer scientist [3]. Besides, it also encompasses the intersection of data analytics and machine learning. Therefore, data science encompasses a heterogeneous group of studies and methodologies such as big data, machine learning, data analytics, and statistics, which challenge the implementation of a single platform capable of incorporating all the available techniques.

There are a variety of widely adopted platforms available for data analysis and knowledge extraction, for example, Tableau,[1] Dataiko,[2] Microsoft Azure Machine Learning Studio,[3] Orange BioLab,[4] each one suitable for a specific step of a data science process. A workflow can be formulated based on the coordinated application of different tools to extract knowledge from massive datasets. In this context, the use of cloud platforms for data science steadily grows because they offer scalability and distributed execution of individual tasks.

In data science, a large dataset allows the generation of a more in-depth model, which provides more robust insights because there are more instances to compose the statistical analysis of data. One of the most relevant aspects regarding a dataset is the quality of available data. Thus, before the use of any statistical method, the

[1] https://www.tableau.com/

[2] https://www.dataiku.com/

[3] https://azure.microsoft.com/en-us/services/machine-learning-studio/

[4] https://orange.biolab.si/

dataset must go through a cleaning process that ensures the uniformity of values and the elimination of duplicated data. On one hand, a large dataset with high-quality data enables an insightful model. On the other hand, the computational power required to process data is directly proportional to the size of the available dataset. In this scenario, high-performance computing (HPC) provides the infrastructure (clusters, grids, and cloud platforms) required to optimize the processing time of data science workflows. In particular, data science demands are transformed in a collection of tasks, with or without the notion of dependency among them, which must be efficiently scheduled along the computational resources [memory, processors, cores, cluster nodes, graphical processing unit (GPU) cores, grid nodes, and virtual machines, for example] to provide the results in an acceptable time interval. To map such tasks to resources, a scheduling policy takes place where load balancing algorithms are important to provide a better execution equilibrium among the tasks and a fast response time, mainly when considering either dynamic or heterogeneous environments. While some articles explore the use of HPC for data science tasks [4–6], in the best of our knowledge, there are no studies that conduct an in-depth analysis of how the aspects of scheduling and load balancing affect data science workflows.

Hence, the present book chapter proposes an analysis of scheduling and load balancing from the perspective of data science scenarios. Furthermore, it presents concepts, environments, and tools for data science to summarize the theoretical background required to understand the definition and execution of data science workflows. Even though its focus lies on presenting concepts, the chapter also illustrates new trends concerning the intersection of data science, scheduling, and load balance.

The remainder of this chapter is organized as follows: Section 2 presents an in-depth explanation of concepts, workflow, problem classes, and tools used by data science. Section 3 explores scheduling and load balancing as tools to leverage the computational power required by data science applications. Section 4 points to open challenges and trends in the use of HPC applied to data science problems. Finally, Section 5 concludes the chapter with closing remarks and directions for future work.

2. Fundamental concepts

This section presents the fundamental concepts related to data science. These are key to understand the concept of HPC, more specifically scheduling and load balancing, impact data science processes, as discussed later in the chapter. The remainder of the section discusses the fundamental components of a data science pipeline, as observed in real-world scenarios.

2.1 Data science workflow

Data science is highly dependent on its application domain and employs complex methods. Nevertheless, it has a very organized pipeline, which varies in the number of steps required to extract knowledge. Current work explores a pipeline that varies between five and seven steps, but in all cases, the process yields similar outputs. This section aims at presenting the most complete process, composed of seven steps, widely used by both companies and researchers. **Figure 2** depicts the flow of information step by step. Moreover, the seven proposed steps can be enumerated as:

1. business understanding;
2. data extraction;
3. data preparation;
4. data exploring;
5. data model;
6. results evaluation; and
7. implementation.

Step 1 refers to the process of understanding in which context the data are inserted on, and what is the expected output. This is a high time-consuming process in a project. However, data scientist must have a deep understanding about the application domain to validate the model's structure as well as its outputs. After understanding the scope of the project, on Step 2, exploring the data that correlate with the problem understood in the last step. These data can be hosted at the client or not. If the client does not have useful data available, the data scientist must look for a synthetic or publicly available dataset to extract the knowledge. Furthermore, on Step 3, techniques are employed to clean data because there is a high chance that it is unorganized or unreadable, so it is necessary to preprocess and standardize it. An example of this step is a dataset that has a column with country names, but in some registers, the value of this column is "Brasil" and in others, it is "Brazil," both values symbolize the same information but are encoded in different languages. Regarding Step 4, the data are organized, and it is indispensable to execute a

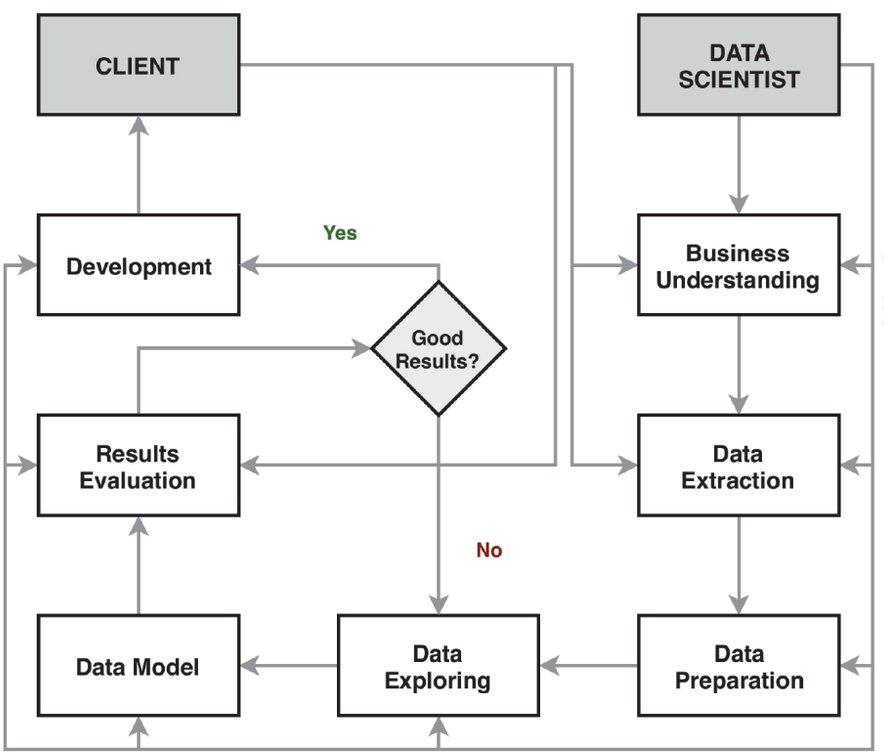

Figure 2.
Flow diagram of data science steps.

detailed analysis in order to figure out patterns or insights that would be valuable to the client. In this stage, the data scientist usually uses plotting techniques to make the data more readable and figure out information.

On Step 5, previously identified insights serve as input. But at this step, it is vital to fully understand the data since, without formal knowledge, it is very hard to fit a model that correctly represents it. At this stage, it is required that the data scientist uses computer science expertise to choose the better approach to plan and validate the model. In Step 6, outputs generated by the model are evaluated in order to analyze how useful they actually are. Usually, this evaluation is conducted by the client and the data scientist together, to examine graphs, numbers, and tables and define if the model generated acceptable results. Finally, on Step 7, the results are validated, and the model is ready to be implemented and deployed in production; thus, it is applied to prediction tasks, having real data as input. The architecture used to implement the model is very important, that is, it is necessary to understand what will be used for the client application since if the client needs a real-time response, the structure will be very different than a nonreal-time scenario.

2.2 Solving problems with data science

Data science is not exclusively employed in business scenarios, and it can be generalized to a plethora of applications, such as in Obama's campaign for US presidential elections in 2012. In the context of this election, technology was applied to identify who were the voters that should receive more attention and marketing influence. Some analysts highlighted the use of data science as fundamental to Obama's victory. However, data science is not limited to the analysis of scenarios, such as in the above example. Many other challenges can benefit from solutions based on data science methodologies. For instance, some problem classes in data science are pattern detection, anomaly detection, cleaning, alignment, classification, regression, knowledge base construction, and density estimation [7]. These classes of problems are explored next.

2.2.1 Pattern detection

The patterns existing in a dataset are not always easily identifiable due to the organization of the data. It is the method employed to discover information standards in the dataset. Figuring patterns hidden into data is a relevant task in several scenarios, for example, to join the clients with similar characteristics, such as those with the same taste or opinion.

2.2.2 Anomaly detection

The distribution of a dataset regards the positions of data points among each other in the dataset. Usually, the representation of this distribution employs a Cartesian plane, in which each point is an instance of the dataset. Within this representation, regions with a defined concentration of data points become clusters. Therefore, outliers are the data points that are too far from these clusters. Anomaly detection aims to classify each data into a dataset as an outlier or not. For example, the banks employ this approach in the scenario of fraud detection. In this scenario, transactions of a client become clusters, and a new transaction is considered as unclassified data.

2.2.3 Cleaning

Dataset contents can present a broad variety of formats, for example, dates, numeric values, and text data. In many cases, values may also be differently formatted across dataset entries, even for the same field, due to human error or lack of standardization. This situation incurs in errors in the dataset, resulting in possible information loss when processing it. The best solution to this problem is to employ methods that manage the dataset contents and standardize the data. For example, in a dataset containing clients birthdays, it is possible that a user fills the information with an invalid date. This case results in information loss because the analysis cannot extract useful information related to the user's birthday.

2.2.4 Alignment

An organized and standardized dataset is fundamental for the generation of trustworthy outputs from data science processes. The process of data alignment is an essential step in dataset standardization. It involves updating the dataset to avoid the use of multiple values to represent the same information. For example, in a dataset containing a gender field, users may use both "M" and "Male" values to represent the same gender. In this context, it is fundamental to unify both entries in one because the information in both is the same.

2.2.5 Classification

Classification regards assigning specific labels to entries in a dataset. A label is any information that presents a limited scope of possibilities, for example, a dozen options present in a specific dataset field. Examples of labels include sentiments, states, and the scale of integer numbers. This dataset field can then be used to group multiple dataset entries according to the unique values that the label may take. For example, in a dataset about client's purchases, each product may be associated with a set of keywords. These keywords can then be used to classify the types of purchases a particular client makes, enabling targeted recommendations for other products.

2.2.6 Regression

Datasets do not always contain fields with labels that enable the classification of data. Nevertheless, in some problems, it is necessary to label values without a restricted group of options, for example, using a field that contains real numbers. This scenario requires a regression approach to estimate which classes an entry should receive, without considering the limited options available in labels. For example, in a dataset with prices and sizes of houses in New York, it is possible to use regression to estimate the price of a new house according to its size. Although the regression problem has an output similar to classification, its output does not have a limited set of values, as occurs in labeling.

2.2.7 Knowledge base construction

Datasets are essential for data science, and the problem of knowledge base construction refers to the process of compiling information to create them. Frequently, this process requires the use of cleaning and alignment methods to

Stage	Tools
Store data	MySql, Mongo DB, Cassandra, PLSql, Redis, HBase
Data preparation	Apache Hive
Data exploring	Knime, Elasticsearch
Data model	Python, R, Julia, Clojure, SPSS, SAS, Apache Manhout
Results evaluation	Tableau, Cognos, ggplot, QlikView, Power BI
Development	Apache Hadoop, Java, Scala, C, Apache Spark, Haskell

Table 1.
Tools used per step of data science.

standardize data. There are a broad group of knowledge bases on the Internet, for example, Kaggle,[5] UCI Repository,[6] Quandl,[7] and MSCOCO.[8]

2.2.8 Density estimation

Density estimation focuses on identifying the clusters that group sets of data points that represent the entries in a dataset. This process is a fundamental step to generate the clusters required by anomaly detection methodologies, as described above. Clustering is another suitable technique to identify groups of entries that may contain related knowledge within a dataset.

2.3 Data science tools

It is difficult to find a tool that fits all data science processes because, as previously mentioned, there are multiple steps with a variety of methods available for use. Hence, there are specific tools for each step, which will provide the most appropriate result. **Table 1** summarizes the most commonly cited tools for each one of the data science workflow steps. The table has a row that is not considered a step of the process, but it is fundamental to results that are Storage Data, which refer to all technologies used for persisting the data in an environment. In the section of data model, some programming languages are cited, but it is hard for a data scientist to employ a language without libraries. For example, using Python is very usual to use libraries such as pandas, sci-kit learn, numpy, and ggplot.

3. Exploring scheduling and load balancing on data science demands

The scheduling problem, in a general view, comprises both a set of resources and a set of consumers [8]. Its focus is to find an appropriate policy to manage the use of resources by several consumers in order to optimize a particular performance metric chosen as a parameter. The evaluation of a scheduling proposal commonly considers two features: (1) performance and (2) efficiency [9]. More specifically, the evaluation comprises the obtained scheduling as well as the time spent to execute the scheduler policies. For example, if the parameter to analyze the

[5] https://www.kaggle.com

[6] https://archive.ics.uci.edu/ml/index.php

[7] https://www.quandl.com/

[8] http://cocodataset.org/home

achieved scheduling is the application execution time, the lower this value, the better the scheduler performance. In turn, efficiency refers to the policies adopted by the scheduler and can be evaluated using computational complexity functions [10].

The general scheduling problem is the unification of two terms in everyday use in the literature. There is often an implicit distinction between the terms scheduling and allocation. Nevertheless, it can be argued that these are merely alternative formulations of the same problem, with allocation posed in terms of resource allocation (from the resources point of view), and scheduling viewed from the consumers' point of view. In this sense, allocation and scheduling are merely two terms describing the same general mechanism but described from different viewpoints. One important issue when treating scheduling is the grain of the consumers [11]. For example, we can have a graph of tasks, a set of processes, and jobs that need resources to execute. In this context, scheduling schemes for multiprogrammed parallel systems can be viewed in two levels. In the first level, processors are allocated to a specific job. In the second level, processes from a job are scheduled using this pool of processors.

We define static scheduling considering the scheduling grain as a task [8]. If data such as information about the processors, the execution time of the tasks, the size of the data, the communication pattern, and the dependency relation among the tasks are known in advance, we can affirm that we have a static or deterministic scheduling model. In this approach, each executable image in the system has a static assignment to a particular set of processors. Scheduling decisions are made deterministically or probabilistically at compile time and remain constant during runtime. The static approach is simple to be implemented. However, it is pointed out that it has two significant disadvantages [11]. First, the workload distribution and the behavior of many applications cannot be predicted before program execution. Second, static scheduling assumes that the characteristics of the computing resources and communication network are known in advance and remain constant. Such an assumption may not be applied to grid environments, for instance.

In the general form of a static task scheduling problem, an application is represented by a directed acyclic graph (DAG) in which nodes represent application tasks, and edges represent intertask data dependencies [12].

Each node label shows computation cost (expected computation time) of the task, and each edge label shows intertask communication cost (expected communication time) between tasks. The objective function of this problem is to map tasks onto processors and order their executions, so that task-precedence requirements are satisfied, and the minimum overall completion time is obtained.

In the case that all information regarding the state of the system as well as the resource needs of a process is known, an optimal assignment can be proposed [11]. Even with all information required for the scheduling, the static method is often computationally expensive getting to the point of being infeasible. Thus, this fact results in suboptimal solutions. We have two general categories within the realm of suboptimal solutions for the scheduling problem: (1) approximate and (2) heuristic. Approximate scheduling uses the same methods used in the optimal one, but instead exploring all possible ideal solutions, it stops when a good one is achieved. Heuristic scheduling uses standard parameters and ideas that affect the behavior of the parallel system. For example, we can group processes with higher communication rate to the same local network or sort works and processors in lists following some predefined criteria in order to perform an efficient mapping among them (list scheduling).

Dynamic scheduling works with the idea that a little (or none) a priori knowledge about the needs and the behavior of the application is available [9]. It is also unknown in what environment the process will execute during its lifetime.
The arrival of new tasks, the relation among them, and data about the target

architecture are unpredictable, and the runtime environment takes the decision of the consumer-resource mapping. The responsibility of global scheduling can be assigned either to a single processor (physically nondistributed) or practiced by a set of processors (physically distributed). Within the realm of this last classification, the taxonomy may also distinguish between those mechanisms that involve cooperation between the distributed components (cooperative) and those in which the individual processors make decisions independent of the actions of the other processors (noncooperative). In the cooperative case, each processor has the responsibility to carry out its portion of the scheduling, but all processors are working toward a common system-wide goal.

Data science comprises the manipulation of a large set of data to extract knowledge [13, 14]. To accomplish this, we have input that is passed through processing engines to generate valuable outputs. In particular, this second step is usually processed as sequential programs that implement both artificial intelligence and statistical-based computational methods. We can take profit from the several processing cores that exist in today's processors to map this sequential demand to be executed in a multithreading program. To accomplish this, Pthreads library and OpenMP are the most common approaches to write multithread parallel programs, where each thread can be mapped to a different core, so exploiting the full power of a multiprocessor HPC architecture.

In addition to multiprocessor architectures, it is possible to transform a sequential code in message passing interface (MPI)-based parallel one, so targeting distributed architectures such as clusters and grids [15]. In this way, contrary to the prior alternative that encompasses the use of standard multiprocessor systems, the efficient use of MPI needs a parallel machine that generally has higher financial costs. Also, a distributed program is more error prone, since problems in the nodes or the network can put all application down. Repairing these future problems sometimes is not trivial, requiring graphical tools to observe processes' interactions. Finally, in addition to multicore and multicomputer architectures, we also have the use of GPU, where graphic cards present a set of nongeneral purpose processors to execute vector calculus much faster than the conventional general-purpose processors [14]. The challenge consists in transforming a sequential code in a parallel one in a transparent way at the user viewpoint, in such a way the data science demand can run faster in parallel deployments. Moreover, the combination of these three aforesaid parallel techniques is also a challenge since optimizations commonly vary from one application to another.

Cloud computing environments today also represent a viable solution to run data science demands [16]. Providers such as Amazon EC2, Microsoft Azure, and Google Cloud have HPC-driven architectures to exploit multiprocessor, multicomputer, and GPU parallelism. In particular, different from standard distributed systems, cloud computing presents the resource elasticity feature where an initial deployment can be on-the-fly changed following the input demand. Thus, it is possible to scale resources in or out (through the addition or removal of containers/virtual machines) or to scale down or up (by performing resource resizing in virtual units) in a transparent way to the user. Logically, the own data science application must be written in such a way to take profit of newly available resources as the current set of working resources.

The CPU load is the most common metric to drive resource elasticity data science demands since most of them execute CPU-bound artificial intelligence-based algorithms. Any network data manipulation through the TCP protocol uses CPU cycles since this is a software protocol executed in the kernel of the operating system and executes software routines to provide data transfer reliability.

Load balancing and resource scheduling are sometimes seen as having the same functionality. However, there is a slight difference: one of the members of resource

scheduling is the scheduling, and this policy can employ or not load balancing algorithms [14]. The basic idea of load balancing is to attempt to balance the load on all processors in such a way to allow all processes on all nodes to proceed at approximately with the same rate. The most significant reason to launch the load balancing is the fact that exists an amount of processing power that is not used efficiently, mainly in dynamic and heterogeneous environments, including grids. In this context, schedulers' policies can use load balancing mechanisms for several purposes, such as: (1) to choose the amount of work to be sent to a process; (2) to move work from an overloaded processor to another that presents a light load; (3) to choose a place (node) to launch a new process from a parallel application; and (4) to decide about process migration. Load balancing is especially essential for some parallel applications that present synchronization points, in which the processes must execute together with the next step.

The most fundamental topic in load balancing consists of determining the measure of the load [13, 15]. There are many different possible measures of load including: (1) number of tasks in a queue; (2) CPU load average; (3) CPU utilization at specific moment; (4) I/O utilization; (5) amount of free CPU; (6) amount of free memory; (7) amount of communication among the processes; and so on. Besides this, we can have any combinations of the above indicators. Considering the scope of processes from the operating system, such measures will influence in deciding about when to trigger the load balancing, which processes will be involved, and where are the destination places in which these processes will execute. Especially on the last topic, other factors to consider when selecting where to put a process include the nearness to resources, some processor and operating system capabilities, and specialized hardware/software features. We must first determine when to balance the load to turn the mechanism useful. Doing so is composed of two phases: (1) detecting that a load unbalancing exists and (2) determining if the cost of load balancing exceeds its possible benefits.

The use of load balancing in data science demands can vary depending on the structure of the parallel applications: Master-Slave, Bag of Tasks, Divide-and-Conquer, Pipeline, or Bulk-Synchronous Parallel [15, 16]. In the first two, we usually have a centralized environment where it is easy to know data about the whole set of resources, to dispatching tasks to them following their load and theoretical capacity. A traditional example of a combination of these parallel applications is the MapReduce framework. In the divide-and-conquer applications, we have a recursive nature to execute the parallel application where new levels of child nodes are created with the upper one cannot execute the tasks in an acceptable time interval. The challenge consists of dividing the tasks rightly following the capacity of the resources. Pipeline-based applications, in their turn, have a set of stages where each incoming task must cross. In order to maintain the cadence between the stages, they must execute in the same time interval, so an outcoming task from the stage n serves as the direct input for the stage $n + 1$. However, the fact of guaranteeing this capacity is not a trivial procedure because of the stages commonly present different complexities in terms of execution algorithms. Finally, bulk-synchronous applications are composed by supersteps, each one with local processing, and arbitrary communication and barrier phases. Load balancing is vital to guarantee that the slowest process does not compromise the performance of the entire application.

4. Open opportunities and trends

This section aims at compiling the previous two sections, so detailing open opportunities and trends when joining resource scheduling and load balancing and the area of data science. In this way, we compile these aspects as follows:

- Automatic transformation of a sequential data science demand to a parallel one—today data science executes locally to query databases and to build knowledge graphs. Sometimes these tasks are time consuming, then it is pertinent to transform a sequential demand in a parallel one to execute faster on multicore, multinode, and GPU architectures.

- Use of GPU cards as an accelerator for data science algorithms—write of data science demands that combine R and Python together with OpenCL or CUDA programming languages, so combining CPU and GPU codes with running fast and in parallel to address a particular data science demand.

- Combination of multimetric load balancing engine to handle data science efficiently—data science typically encompasses excellent access to IO (including main memory and hard disk) and a high volume of CPU cycles to process CPU-bound algorithms. In this way, the idea is to execute data science demands and learn their behavior, so proposing an adaptable load balancing metrics that take into account different parameters as input.

- Task migration heuristics—when developing long-running data science parallel codes, it is essential to develop task migration alternatives to reschedule demands from one resource to another. This is particularly pertinent on dynamic environments, either at the application or infrastructure level.

- Cloud elasticity to address data science demand—cloud elasticity comes to adapt the number of the resource following the current demand. Thus, we propose a combination of vertical and horizontal elasticity, together with reactive and proactive approaches to detect abnormal situations. We can use both consolidation and inclusion of resources, aiming to always accommodate the most appropriate number of resources for a particular and momentaneous data science demand.

- Definition of a standard API to deal with data science—frequently enterprises present several departments, each one with its data science demands. In this way, we envisage an opportunity on developing a standard framework (with a standard API too) to support the data science demands of the whole enterprise. The idea is to provide a dashboard with a collection of data science functions, also expressing the expected input and the output for each one.

- Smart correlation of events—enterprises regularly have timed data in several databases. We present an opportunity, at each time a problem is found, to take this particular timestamp and compare in the data sources looking for eventual data correlations. Thus, we can perceive relations such as: (1) if this happens, these other things will also happen and (2) this event happened because a set of prior events happened beforehand.

- Benchmark to evaluate a mapping of data science tasks to HPC resources—how we know if particular scheduling outperforms another one for executing a particular data science demand? We see as an opportunity for the exploration of benchmarks to evaluate scheduling and load balancing techniques that manipulate data science tasks. Thus, such benchmarks must define what they expect as input and provide a set of metrics as output. Yet, the output can be a single value, a collection of values (as a data vector), or a collection of elements of a data structure (e.g., timestamp and data are useful to develop user profiles and tracking of assets).

- Simulation environment to execute data science demands on distributed resources, but doing all of this a local program—simulation environments, like Simgrid or GridSim, are useful to use a sequential program to test and simulate complex parallel demands on a set of virtual resources. Thus, we can save time on testing different parameters and algorithms when developing scheduling and load balancing algorithms for data science.

- Definition of metrics to evaluate the scheduling and/or load balancing of data science tasks—CPU load, memory footprint, disk space, network throughput, and cache hit rate are examples of metrics that are commonly employed on distributed systems. Data science is a new area of knowledge, where we encourage the definition of new metrics to compare the execution of data science demands.

5. Conclusion

The continuous generation of data by different industry segments presents a valuable opportunity for analysis and knowledge extraction through data science methods. There is a high interest in studies that explore the application of data science to a variety of scenarios, each one with distinct characteristics that reflect on the composition of available datasets. Furthermore, there is not a single data science methodology that is applied to all possible data science problems. Consequently, the most common approach to data science problems is to define a sequence of methods that depend on the characteristics of the dataset and the intended results.

The constant growth in dataset sizes and the complexity of specific data science methods also impose a considerable challenge to provide the computational power required to process data and extract meaningful knowledge. In this context, cloud, fog, and grid computing architectures present themselves as ideal solutions to apply data science processes to massively sized datasets.

The distributed nature of such environments raises a series of new challenges, some of which widely studied in the literature. Nevertheless, the unique characteristics of data science workloads bring new aspects to these challenges, which require renewed attention from the scientific community.

This chapter focused on the specific challenge of scheduling and load balancing in the context of computational environments applied to data science. We presented an overview of data science processes, in addition to how scheduling and load balancing methodologies impact these processes and what aspects to consider when using distributed environments applied to data science. In particular, the challenge of enabling the automatic transformation of sequential data science demands into parallel ones is of particular interest because it abstracts part of the complexity involved in parallelizing data science tasks. As a result, such an automatic transformation promotes wider adoption of distributed environments as standard tools for large-scale data science processes.

Another notable challenge is to develop cloud elasticity techniques tailored to data science tasks. Such techniques must consider the specific requirements of data science processes to guarantee the proper reservation of resources and migration of tasks in order to guarantee a high throughput for such scenarios. These and the other investigated challenges represent prime research opportunities to increase the performance of data science processes.

DOI: http://dx.doi.org/10.5772/intechopen.92578

Author details

Diórgenes Eugênio da Silveira[1], Eduardo Souza dos Reis[1], Rodrigo Simon Bavaresco[1], Marcio Miguel Gomes[1], Cristiano André da Costa[1], Jorge Luis Victoria Barbosa[1], Rodolfo Stoffel Antunes[1], Alvaro Machado Júnior[2], Rodrigo Saad[2] and Rodrigo da Rosa Righi[1*]

1 Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, Brazil

2 DELL Computadores do Brasil, Eldorado do Sul, RS, Brazil

*Address all correspondence to: rrrighi@unisinos.br

References

[1] Amirian P, van Loggerenberg F, Lang T. Data Science and Analytics. Cham: Springer International Publishing; 2017. pp. 15-37

[2] Gibert K, Horsburgh JS, Athanasiadis IN, Holmes G. Environmental data science. Environmental Modelling and Software. 2018;**106**:4-12. Special Issue on Environmental Data Science. Applications to Air quality and Water cycle

[3] Grus J. Data Science from Scratch: First Principles with Python. 1st ed. O'Reilly Media, Inc.; 2015. Available from: https://www.amazon.com/Data-Science-Scratch-Principles-Python/dp/149190142X

[4] Ahmad A, Paul A, Din S, Rathore MM, Choi GS, Jeon G. Multilevel data processing using parallel algorithms for analyzing big data in high-performance computing. International Journal of Parallel Programming. 2018;**46**(3):508-527

[5] Bomatpalli T, Wagh R, Balaji S. High performance computing and big data analytics paradigms and challenges. International Journal of Computer Applications. 2015;**116**(04):28-33

[6] Singh D, Reddy CK. A survey on platforms for big data analytics. Journal of Big Data. 2014;**2**(1):8

[7] Dorr BJ, Greenberg CS, Fontana P, Przybocki M, Le Bras M, Ploehn C, et al. A new data science research program: evaluation, metrology, standards, and community outreach. International Journal of Data Science and Analytics. 2016;**1**(3):177-197

[8] Chasapis D, Moreto M, Schulz M, Rountree B, Valero M, Casas M. Power efficient job scheduling by predicting the impact of processor manufacturing variability. In: Proceedings of the ACM International Conference on Supercomputing (ICS'19). New York, NY, USA: ACM; 2019. pp. 296-307

[9] Liu L, Tan H, Jiang SH-C, Han Z, Li X-Y, Huang H. Dependent task placement and scheduling with function configuration in edge computing. In: Proceedings of the International Symposium on Quality of Service (IWQoS'19). New York, NY, USA: ACM; 2019. pp. 20:1-20:10

[10] Palyvos-Giannas D, Gulisano V, Papatriantafilou M. Haren: A framework for ad-hoc thread scheduling policies for data streaming applications. In: Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19). New York, NY, USA: ACM; 2019. pp. 19-30

[11] Feng Y, Zhu Y. PES: Proactive event scheduling for responsive and energy-efficient mobile web computing. In: Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19). New York, NY, USA: ACM; 2019. pp. 66-78

[12] Topcuoglu H, Hariri S, Min-You WU. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems. 2002;**13**(3):260-274

[13] Menon H, Kale L. A distributed dynamic load balancer for iterative applications. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'13). New York, NY, USA: ACM; 2013. pp. 15: 1-15:11

[14] Schepis L, Cuomo F, Petroni A, Biagi M, Listanti M, Scarano G. Adaptive data update for cloud-based internet of things applications. In:

Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era (PERSIST-IoT'19). New York, NY, USA: ACM; 2019. pp. 13-18

[15] Bak S, Menon H, White S, Diener M, Kale L. Integrating openmp into the charm++ programming model. In: Proceedings of the Third International Workshop on Extreme Scale Programming Models and Middleware (ESPM2'17). New York, NY, USA: ACM; 2017. pp. 4:1-4:7

[16] Gandhi R, Liu HH, Hu YC, Lu G, Padhye J, Yuan L, et al. Duet: Cloud scale load balancing with hardware and software. SIGCOMM Computer Communication Review. 2014;**44**(4): 27-38