

Feature Selection using LAID and its Implementation on High-Performance Computing Systems

A parallel computing approach using Python and HDF5

Paulo Pereira Morgado

Thesis to obtain the Master of Science Degree in
Information and Enterprise Systems

Supervisor: Prof. Luís Manuel Pereira Sales Cavique Santos

Examination Committee

Chairperson: Prof. Miguel Leitão Bignolas Mira da Silva

Supervisor: Prof. Luís Manuel Pereira Sales Cavique Santos

Member of the Committee: Prof. Armando Brito Mendes

October 2021

Abstract

Of the many current challenges that the increasing availability of data poses to society and organizations, the ability to collect, store, process, analyze and extract knowledge in a helpful time frame stands out.

This dissertation focuses on the processing of highly dimensioned datasets, such as those generated by High-throughput sequencing (HTS) techniques, which are increasingly common in areas of knowledge such as bioinformatics.

Among the dimensionality reduction techniques, feature selection has become crucial since it reduces the high dimensionality of large datasets.

A possible approach to perform the feature selection that considerably reduces the dimension without increasing the inconsistency of the data is the use of Logical Analysis of Inconsistent Data (LAID). Several studies in recent years have demonstrated its potential in solving this problem and highlighted its advantages as a systematic methodology, robust, easy to interpret, and capable of dealing with inconsistent data, (Cavique *et al.*, 2018), (Apolónia and Cavique, 2019a).

The same studies revealed processing times longer than desired for full utilization and pointed out the solution for executing the algorithms using parallel processing and mobilization of a high-performance computing (HPC) installation.

This work represents another contribution to this effort by addressing dataset storage methods suitable for parallel processing, algorithm parallelization solutions, high-performance environment configuration. Finally, it tests in the HPC environment of the Infraestrutura Nacional de Computação Distribuída (INCD). What allowed us to describe a fit-for-purpose dataset storage solution as well as parallel processing in an HPC environment can reduce processing times for the end-user, achieving a satisfactory 82% reduction relative to best sequential time.

Keywords: Data mining, High dimensionality, Feature selection, Logical Analysis of Inconsistent Data (LAID), Parallel computing

Resumo

Dos muitos desafios atuais que a crescente disponibilidade de dados lança à sociedade e as organizações destaca-se a capacidade de coletar, armazenar, processar, analisar e extrair conhecimento em tempo útil.

Nesta dissertação o foco é o processamento de *datasets* altamente dimensionados, como os gerados pelas técnicas de *High-throughput sequencing* (HTS), cada vez mais comuns em áreas do conhecimento como a bioinformática.

De entre as técnicas de redução de dimensionalidade a seleção de atributos tornou-se crucial ao permitir reduzir a alta dimensionalidade de grandes quantidades de dados, que sem esse tratamento permaneceriam com utilidade limitada.

Uma possível abordagem para realizar a seleção de atributos que permite reduzir consideravelmente a dimensão sem aumentar a inconsistência dos dados é a utilização de Análise Lógica de Dados Inconsistentes (LAID). Vários estudos nos últimos anos, demonstraram as suas potencialidades na resolução deste problema e evidenciaram as suas vantagens como uma metodologia sistemática, robusta, de fácil interpretação e capaz de lidar com dados inconsistentes, (Cavique *et al.*, 2018), (Apolónia and Cavique, 2019a).

Os mesmos estudos revelaram tempos de processamento acima do desejado para uma utilização plena e apontaram a solução para a execução dos algoritmos usando processamento paralelo e a mobilização de recursos de uma instalação de computação de alto desempenho (HPC).

Este trabalho representa mais um contributo nesse esforço ao abordar formas de armazenamento dos *datasets* com vista ao seu processamento paralelo, soluções de paralelização dos algoritmos, configuração do ambiente de alto desempenho e finalmente os testes no ambiente HPC da Infraestrutura Nacional de Computação Distribuída (INCD). O que permite descrever uma solução de armazenamento do *dataset* adequada à finalidade, bem como o processamento paralelo em um ambiente HPC, pode reduzir os tempos de processamento para o utilizador final, alcançando uma redução satisfatória de 82% relativa ao melhor tempo sequencial.

Palavras-chave: Data Mining, Grande número de dimensões, Seleção de atributos, Análise lógica de dados inconsistentes (LAID), Computação paralela

Acknowledgments

I would like to take this opportunity to express my gratitude to those who directly or indirectly collaborated in the development of this dissertation

First, I would like to express my sincere appreciation to my Supervisor, Professor Luis Cavique for the challenge and opportunity to carry out this experience in an HPC environment as well as for his wisdom, support, and patience. His guidance was essential for the outcome.

I also want to thank all the help and support provided by the excellent INCD team, especially Mário David, João Pina, and João Martins. Without you, this work would not have been possible.

It's impossible not to thank the faculty and all my colleagues at MISE, especially Mauro Viegas, Joaquim Santos, and Fernando Diogo, regular co-workers. May the force be with you.

I also thank my brothers António and Fábio for their patience and unconditional support throughout this journey.

Finally, a very special thanks to my good friend Sónia Domingues for her support and advice.

This work was produced with the support of Infraestrutura Nacional de Computação Distribuída - INCD [National Infrastructure for Distributed Computation] funded by FCT and FEDER under the project 01/SAICT/2016 nº 022153

Table of contents

1. Introduction.....	1
1.1 Motivation.....	2
1.2 Scope.....	3
1.3 Research Methodology	5
1.3.1 General	5
1.3.2 The framework: design science research.....	6
1.4 Document Structure	7
2. Theoretical Background.....	8
2.1 Systematic Literature Review.....	9
2.1.1 Research questions.....	9
2.1.2 SLR Planning and Execution	9
2.1.3 SLR contributes towards answering the research questions.	12
2.1.4 Related Work	17
3. Feature Selection with Logical Analysis of Inconsistent Data (LAID).....	19
3.1 Feature Selection	20
3.2 Feature Selection Algorithms in the genesis of LAID	21
3.2.1 Rough Sets	21
3.2.2 LAD.....	22
3.3 Logical Analysis of Inconsistent Data - LAID	22
3.3.1 LAID Steps and Algorithms.....	23
4. Parallel computation and HPC.....	25
4.1 Parallel computing	26
4.2 High-performance Computing - HPC	27
4.3 Infraestrutura Nacional de Computação Distribuída	27
4.3.1 Services provided.....	28
4.3.2 HPC facilities at INCD	28
4.4 HPC Environment for the Experiment.....	28
4.4.1 The Foundations	28
4.4.2 The building blocks	31
4.5 Discussion	34
5. Parallel Code.....	35
5.1 Parallelism in Python.....	36
5.2 Design Methodology for parallel program.....	36
5.3 Coding the LAID algorithms.....	37
5.3.1 Laid steps	37
5.3.2 Step 1 - Check and fix redundant or inconsistent observations.....	37
5.3.3 Step 2 - Disjoint Matrix Generation	38

5.3.4	Step 3 - Find the solution with the minimum set covering algorithm.....	40
5.3.5	Flow control flow and orchestration.....	40
5.4	Performance Metrics for Parallel Systems	41
6.	The Dataset used.....	42
6.1	Assembly the dataset parts in HDF5	44
6.2	Loading the data.....	44
6.3	A brief exploratory data analysis	46
6.4	Discussion.....	48
7.	HDF5 Storage Layout: Chunking Analyses.....	49
7.1	Storage layout options	50
7.1.1	Tests performed	51
7.1.2	Write Operations over the HDF5 Source-Dataset.....	51
7.1.3	Read Operations over the HDF5 Source-Dataset.....	53
7.1.4	Read and write operations over the DM-Dataset.....	54
7.2	Strategy for HDF5 files and datasets	54
7.3	Attempts to reduce the disk size of the DM-Dataset	55
7.3.1	Dataset compression	56
7.3.2	Remove empty columns, the "redux" approach.....	57
7.3.3	The data type option	57
7.4	Discussion.....	58
8.	Computational Experiments and Results	59
8.1	Method and Action Plan.....	60
8.2	Serial, Hybrid and Parallel Runs.....	61
8.3	Serial Program Model.....	62
8.4	Hybrid of Serial and Parallel Programming Models.....	62
8.5	Parallel decomposition through vertical data partition	63
8.6	Discussion.....	64
9.	Conclusions.....	66
9.1	Contributions.....	67
9.2	Limitations.....	68
9.3	Future Work	68
10.	References.....	69
11.	Annex.....	76
11.1	Research align with the three Hevner cycles: Relevance, Design and Rigor	77
11.2	The Artifact.....	78
11.2.1	Configuration file	78
11.2.2	Python Code	79
11.3	Sample Output	80

List of Figures

Figure 1-1 - The Fourth Paradigm	2
Figure 1-2 – A paradigm shift in scientific environments	4
Figure 1-3 - Design Science Research Methodology	6
Figure 1-4 - Major topics of this work.....	7
Figure 2-1 - SLR Review Protocol	10
Figure 2-2 - Distribution of the papers screened and selected over the years.....	11
Figure 2-3 - Schematic of the HDF5 software architecture	15
Figure 3-1 - Categorization of dimension reduction methodologies	20
Figure 4-1 - Flynn's taxonomy for computer architectures	26
Figure 4-2 - Technology usage of Cirrus-A HPC Cluster and components used.....	29
Figure 4-3 - Application cooperation	31
Figure 4-4 - Difference between NumPy Array and Python List	33
Figure 5-1 - DM-Dataset generation, first trial of serial version against 5 parallel tasks	38
Figure 6-1 - Fragment of one of the text files	43
Figure 6-2 - Assembly of the dataset parts in HDF5 dataset	44
Figure 6-3 - Work packages described on the config file.....	45
Figure 6-4 - Distribution of the sum of values in column.....	47
Figure 6-5 - Distributions patterns found on Source-Dataset.....	47
Figure 7-1 - Read operations versus HDF5 Storage layout (time in seconds).....	53
Figure 7-2 - Best fit chunk and contiguous layout comparison for reading workload in seconds.....	54
Figure 7-3 - Disk size estimate for DM-Dataset	55
Figure 7-4 - DM-Dataset generation with and without compression	56
Figure 7-5 - HDF5 Datatype Classes and their properties	57
Figure 8-1 - Bash script for launching jobs through mpirun	60
Figure 8-2 - Serial, Hybrid and Parallel Runs	61
Figure 8-3 - DM-Dataset generation, Comparison of the serial version against 5 parallel tasks.....	62
Figure 11-1 - config.json	78

List of Tables

Table 2-1 - SLR Results	11
Table 2-2 - Publication Type of the selected papers	11
Table 6-1 - Rows, Columns, Classification, and size on disk of Dataset parts	43
Table 6-2 - Time for loading all text files, performed by two parallel tasks	45
Table 6-3 - Data loading into HDF5, Speedup, and Efficiency Estimated	46
Table 6-4 - Sum of values per column vs number of columns	46
Table 7-1 - Datasets used for read/write tests with size on disk	51
Table 7-2 - How is the time consumed?.....	51
Table 7-3 - Chunking layout tested	51
Table 7-4 - Best Fit chunk estimation per dataset	52
Table 7-5 - Average time to write an entire block of data on HDF5 file in seconds	52
Table 7-6 - Read operations versus HDF5 Storage layout (time in seconds)	53
Table 8-1 – Average time spent running Serial LAID steps by step	62
Table 8-2 - Time spent running Hybrid approach	63
Table 8-3 - Performance Metrics for Hybrid approach	63
Table 8-4 - Time spent running Parallel Decomposition	63
Table 8-5 - Performance Metrics for Parallel over Serial	64
Table 8-6 - Comparative board of three approaches Serial/Hybrid/Parallel decomposition	64
Table 11-1 - Research align with Hevner cycles, Relevance, Design and Rigor	77

1. Introduction

1.1 Motivation

Our lives, as individuals and collectives where we belong, as well as our way of life in society, is increasingly driven by data (Pentland, 2013), and dependent on our capacity to collect, store, process, analyze and extract knowledge from a constantly increasing amount of data in a useful time.

The same phenomenon can be observed in parallel in the scientific world (J. Zhang *et al.*, 2020). The exponential growth of data generated by scientific experiments, instruments, and sensors, considering their volume, complexity, and even scale of distribution, has become a critical factor in several science disciplines, which induces the need to analyze ever-increasing volumes of data on time. Some authors identified this need as a new paradigm in scientific production and adopted the term coined by Jim Gray, “The Fourth Paradigm” (Hey, Tansley and Tolle, 2009). According to them, Science started empirically, then theoretical, in the last few decades, computational and, recently, became data-driven. Presented in Figure 1-1 from (Agrawal and Choudhary, 2016), these authors call this trend a “Data-intensive science”, consisting of three basic activities: capture, curation, and analysis.

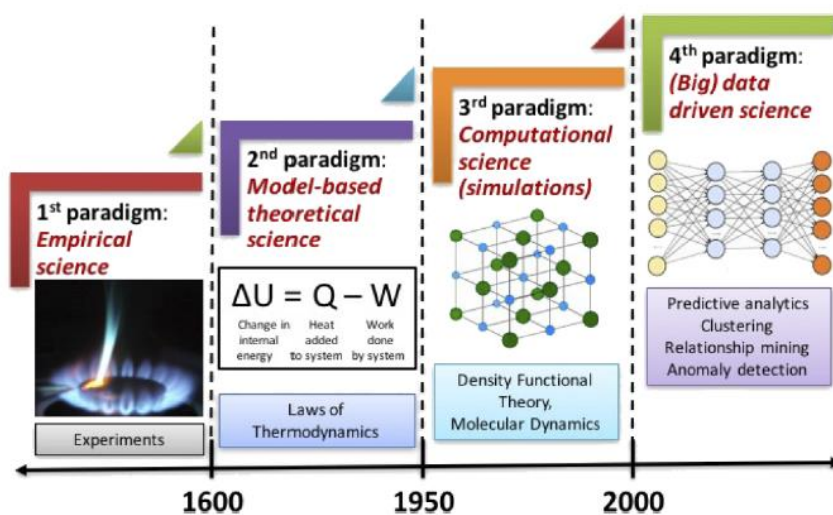


Figure 1-1 - The Fourth Paradigm

Plenty of examples can be chosen to illustrate this trend, considering particles collide in Large Hadron Collider (LHC) detectors approximately 1 billion times per second, generating about one petabyte of collision data per second. However, such quantities of data are impossible for current computing systems to record, and they are hence filtered by the experiments, keeping only the most “interesting” ones. Even after the drastic data reduction performed by the experiments, the CERN DC processes on average one petabyte (PB) of data per day (Gaillard, 2017). Also, the next-generation nuclear fusion device, ITER’s individual experimental runs will have a much longer duration than the current generation of tokamaks and will generate up to 0.4 petabytes (PB) of data per day (Brett, 2016).

Whether in the scientific, business, or the governmental sphere the same need led to the creation of new ways to manage distributed data, processing it, and mainly analyzing it to obtain information and useful knowledge.

This processing and analysis are usually called knowledge discovery from data or KDD and one essential stage of the KDD process (Azevedo and Santos, 2008) is usually known as Data mining that can be defined as the process of discovering patterns and extracting knowledge from large amounts of data (Han, Kamber and Pei, 2012). Patterns on data is a generic expression that aggregates concepts like identifying groupings of data not previously known, detecting unusual records, or finding unknown dependencies. This process can be automatic or, very often, semi-automatic (Witten *et al.*, 2017).

As it is often impracticable to analyze the entire Dataset, it is necessary to know how to choose which features the analysis should focus on, to the detriment of all others. It is on this scope that this work focuses, specifically in situations where this feature selection becomes essential, as in the case of highly dimensioned data sets, as the called Omics datasets. Omics is a neologism associated with biology suffixes such as genomics, transcriptomics, proteomics, or metabolomics (Cavique *et al.*, 2018) and usually refers to biological parameters commonly called biomarkers.

Different types of technologies can be used in functional genomics experiments ranging from real-time PCR, nowadays so used, due to the COVID-19 pandemic to high-throughput technologies such as microarrays and next-generation sequencing (NGS). The output of these technologies denotes a "great challenge for computational techniques, because of their large dimensionality (up to several tens of thousands of genes) and their small sample sizes" (Saeys, Inza and Larrañaga, 2007).

The potential of these technologies is huge "This high-throughput DNA sequencing technology can sequence an entire human genome within a few hours at a cost of just around one thousand US dollars (USD). Only 18 years ago, this feat took the International Human Genome Sequencing Consortium 13 years and three billion USD to accomplish using Sanger sequencing" (Savelieff, 2019).

All these technological advances lead to the creation of more data, greater responsiveness, lower costs, massification of use, probably personalized medicine, and more data to handle. It is in one of these examples that this work focuses on.

1.2 Scope

In data mining one of the relevant tasks is dimensions reduction. One possible way to accomplish it is through feature selection, which is mandatory on highly dimensioned datasets for which the analysis of all the dimensions is impractical and counterproductive. This task becomes even more necessary when the number of observations is low, the data matrix is sparse, and the analysis of row classification reveals inconsistencies.

Because the negative impact of high dimensionality and a low number of observations on learning models is well known, what Richard Bellman called "the curse of dimensionality", (Verleysen and François, 2005) these authors also remind us that "the number of learning data should grow exponentially with the dimension", the

reduction of dimensions is, therefore, a way to obtain the proper proportion between the number of observations and dimensions.

The problem addressed in the present work is an example of that. Considering that a feature selection problem with a dataset involving one million attributes was solved in (Cavique *et al.*, 2018), using partitions of the problem and a new method called Logical Analysis of Inconsistent Data (LAID) proposed by (Cavique, Mendes and Funk, 2011), this present research aims to continue this effort.

This method arises from the combination of Rough Sets Theory (Pawlak, 1991) and Logical Analysis of Data (LAD) methodology (Boros *et al.*, 1997). Both approaches are a subset of filter models, which goal is to reduce the number of dataset attributes using two phases: problem transformation and optimization. Their specificity is to keep the semantics of the data by removing only the redundant data.

According to the author (Cavique, Mendes and Funk, 2011), LAID summarizes the best qualities of each of these theories: being able to deal with inconsistent data and non-dichotomized classes, characteristics of Rough Sets, (Rissino and Lambert-Torres, 2009) as well as the computational effectiveness of LAD.

The 2018 experiment (Cavique *et al.*, 2018) uses a computer cluster running at the Portuguese National Infrastructure for Distributed Computing (INCD), for parallel processing while also using the dataset in memory. As a final remark, the authors consider three major aspects: data, algorithm, and computer environment. Since large datasets with many dimensions are becoming the new normal, they advocate that:

- In-memory data access should be replaced by on-disk data access.
- The new algorithms need low time complexity and problem decomposition to allow parallelizing the problem. Therefore, the intense use of computationally expensive algorithms, such as metaheuristics, should be reduced.
- And last, the desired environment should be in the cloud, running the programs in parallel in High-Performance Computing (HPC).

This is in line with the new trends for compute-intensive algorithms (Talia, 2019). and constitutes a paradigm shift in scientific environments, (Cavique *et al.*, 2018), illustrated in Figure 1-2

	datasets	algorithms	environment
before	in-memory	sequential	workstation
now	on-disk	parallel	cloud

Figure 1-2 – A paradigm shift in scientific environments

Later (Apolónia and Cavique, 2019a) revisited this problem again using LAID and a different computational approach: sequential processing on a single machine and the introduction of a major difference which was that the dataset was stored in high-performance data management and storage suite (HDF5) on disk.

Both experiments concluded that the high processing time remains an additional problem that limits its application.

The challenge and motivation for this new approach to the described problem is to reduce the processing time, using the computational paradigm: parallel processing, on-disk data access, and cloud/HPC environment.

The purpose of this research is to discover how parallel processing can reduce processing times for the end-user and describe how this computational paradigm applies to this problem.

1.3 Research Methodology

1.3.1 General

The choice for the research methodology fell upon Design Science Research Methodology (DSRM), a scientific methodology applied in problem-solving, with increasing use in Information Systems research (Hevner *et al.*, 2004), since it is a pragmatic methodology, aware of the practical utility of the "outcome" and indicated for applied science.

This methodology fits in what the author calls doubt-propelled curiosity, in which the acquisition of a new piece of knowledge instigates new doubts, hence more curiosity, in an infinite cycle, whose only brakes are the scope of this work and the physical limits of these pages, but which undoubtedly persist beyond them.

Through DSRM, new knowledge arises from the creation and evaluation of innovative artifacts, (constructs, models, methods, and applications) or processes, that address the research challenges and constitutes a solution to a relevant and real-world problem.

According to (Hevner, 2007) Design Science Research is structured in three interactive cycles:

- Relevance Cycle, which refers to the environment, namely the domain comprising people, organization, and technological systems, as well as problems and opportunities
- Design Cycle: in charge of the core activities of construction and evaluation of the Artifacts / Processes developed in the research
- Rigor Cycle: which associates activities and conclusions with scientific foundations, state-of-the-art experience, and knowledge

Recognizing the potential of this methodology in IS research, (Hevner, 2007) also proposes a set of seven guidelines to support its application: Design as an artifact, the relevance of the problem, the evaluation of the

design, the research contributions, the rigor of the Research, the design constituting itself as a research process and finally communication of the research and its results.

A related work (Peffer *et al.*, 2007) proposed, with examples, how DSRM can be operationalized in six steps; Problem identification and motivation, Defining the objectives for a solution, Design and development, Demonstration, Evaluation, and finally Communication. These six steps will be used as guidelines for this research.

1.3.2 The framework: design science research

Annex 11.1 presents how the work packages of this research align with the three cycles; Relevance, Design and Rigor and guidelines described by (Hevner, 2007) and also with the six-stage process of DSR preconized by (Peffer *et al.*, 2007)

In this work, the artifact is constituted by:

- A model and a set of guidelines of the best suited and optimized way of storing the dataset on the HDF5 aiming parallelization.
- All the code of the parallelized version of the algorithms constituting LAID.
- The processing orchestration methods.

The entry point for this study is a Design & Development centered approach illustrated in Figure 1-3 adapted from (Peffer *et al.*, 2007).

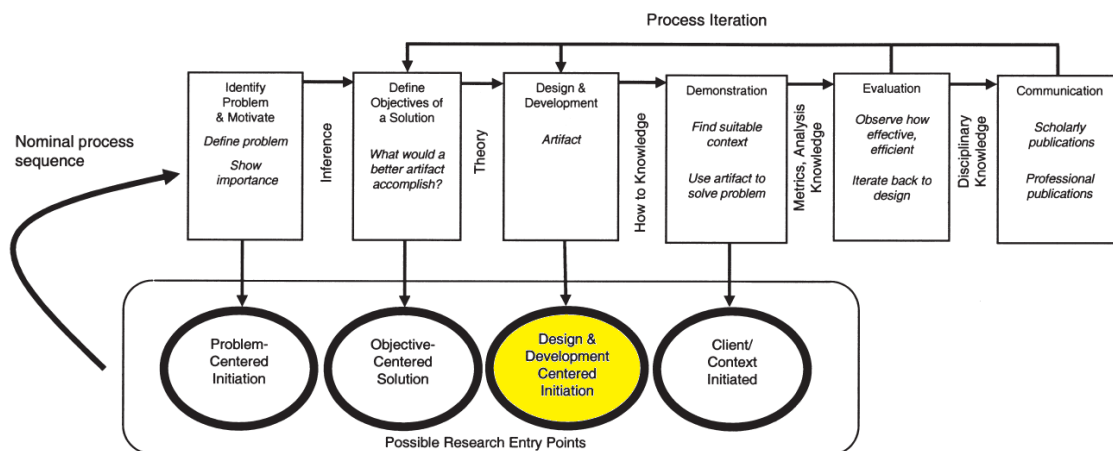


Figure 1-3 - Design Science Research Methodology

1.4 Document Structure

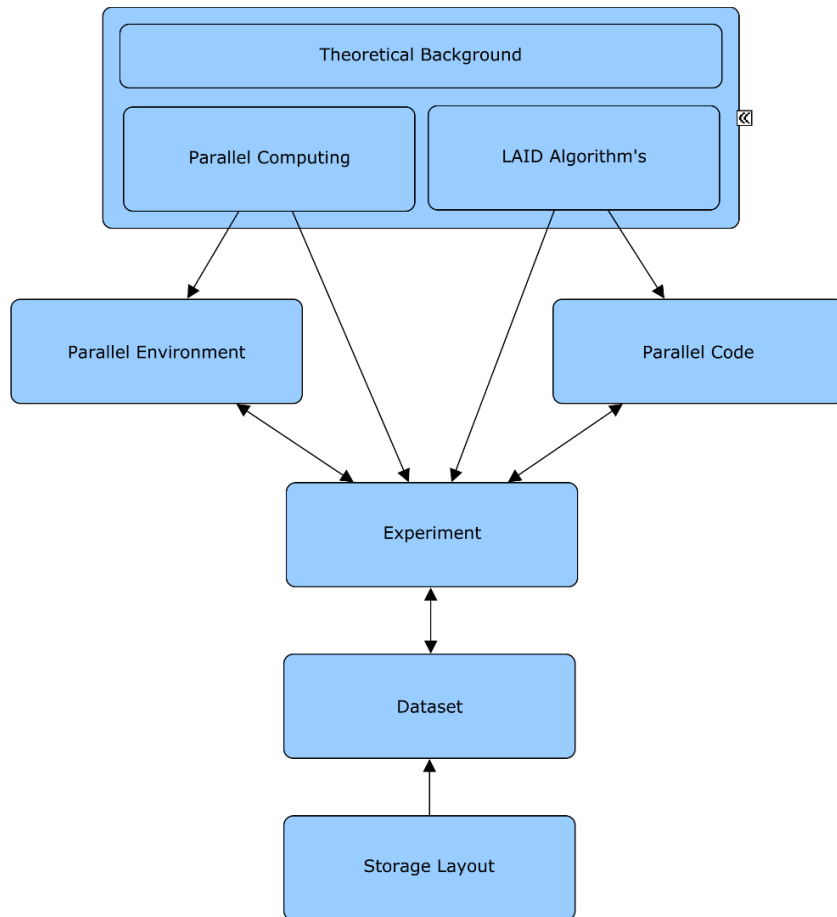


Figure 1-4 - Major topics of this work

Figure 1-4 illustrates the major topics of this work described on the remainder of the document and structured as follows:

Section 2 contains the theoretical background as an outcome of the execution of the Systematic Literature Review, including related work. Section 3 introduces the Logical Analysis of Inconsistent Data (LAID) methodology and its framing in the Feature Selection techniques. Parallel computation and HPC are presented in Section 4 which also includes a detailed description of all the “pieces” of the HPC environment used in this work. Section 5 presents the Parallel Code of LAID’s algorithms used. Section 6 introduces The Dataset used in the computational experiment and in Section 7 the analysis performed on dataset storage options in HDF5 format files is described. Section 8 contains the Computational Experiments and Results and finally, Section 9 presents the Conclusions.

2.Theoretical Background

2.1 Systematic Literature Review

To compile a theoretical background that summarizes and mobilizes existent knowledge about the main goal of this work and to answer the research questions, a systematic literature review (SLR) was conducted, This was performed according to the guidelines proposed by Kitchenham's (Kitchenham, 2004): planning the SLR (phase one), execution (phase two), and result analysis (phase three).

2.1.1 Research questions

To achieve this purpose, the following research questions (RQ) were formulated:

- RQ1. What is the most suitable computational paradigm for this problem?
- RQ2. Is the HDF5 format suitable for use in distributed computing and parallel processing environments?

The final development is expected to result in a kind of service where end users can load and parameterize their own problems and receive results in an acceptable timeframe. So, the main goal of the present research project is to find out how this will be possible and build it.

Though this is a well-defined and delimited problem, the search for potential solutions covers some disciplines of computer and data sciences, therefore, the final contribution of this effort will not only be the final development but also the lessons learned, and all the knowledge obtained during the process.

2.1.2 SLR Planning and Execution

In this planning phase, a review protocol was defined with a focus on research in scientific databases, and not on specific books or technical reports. The following sources were elected to run the SLR research process:

- IEEE Xplore (<http://www.ieee.org/web/publications/xplore/>)
- Science Direct – Elsevier (<http://www.elsevier.com>)
- Springer Link (<http://www.springerlink.com>)
- Sage Journals (<https://journals.sagepub.com>)
- Web of Science (<https://mjl.clarivate.com/>)
- ACM Digital Library (<https://dl.acm.org/>)

The following search keywords were used to find relevant papers in the title, keywords, or abstract: *large dimensions datasets, feature selection, logical analysis inconsistent data, parallel hdf5 io, parallel, grid, computing, evaluate.*

Given the high number of results obtained with some of the keywords while comparing them with the scarce number of results in others and the broad scope of our research questions, it was necessary to carry out several separate searches, using different sets of keywords to maximize the relevant results. As an example, “hdf5”

AND (“parallel” OR “grid”) AND “computing”. On the other hand, the list of keywords grew with the analysis of the preliminary results, which required new searches containing the new keywords.

Inclusion and exclusion criteria were considered. The purpose of these criteria is to facilitate the discovery of studies that are relevant to this research. The main option was to focus on full articles from peer-reviewed journals, conferences, workshops, and book chapters. Avoided masters and doctoral theses.

The option of not defining an interval of years of publication was also followed, assuming that implementing this limit could exclude relevant studies for this work.

In situations where the same content is captured in different versions, we chose to include the most complete version of the study. Below is a summary of the inclusion and exclusion criteria:

Inclusion criteria:

- English, French, Spanish, or Portuguese peer-reviewed studies that answer the research questions.
- Studies with focus on Parallel computing scenarios, with emphasis on scientific experiments including measurement and evaluation of results.
- Studies that focus on the use of parallel disk-based file formats such as HDF5.

Exclusion criteria:

- Studies in a language other than English, French, Spanish, or Portuguese.
- Studies where it is not possible to access the full text.
- Studies that are not related to the research questions.
- Duplicated studies

For the selection of studies, after applying keywords in databases searches to identify candidates, the exclusion criteria were applied based on the analysis of their titles and abstracts, and, next, they were evaluated based on the reading the full text as to its real relevance to this work. as illustrated in Figure 2-1.

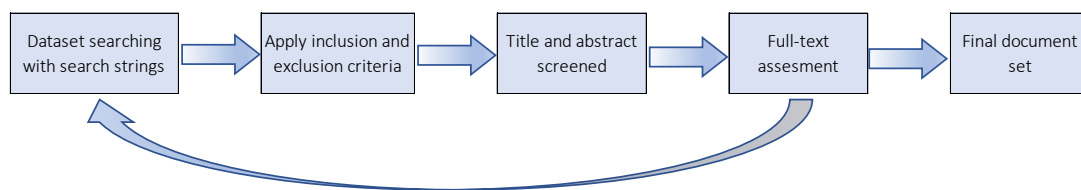


Figure 2-1 - SLR Review Protocol

The feedback collected during the execution of the protocol was also used to improve and refine the search, new terms e.g. *HPC*, *MPI*, *NetCDF*, have been added to the keywords list.

It should also be noted that the quality assessment of the selected studies was not carried out, as if they were approved in the peer review and managed to be published, it was considered that this is sufficient evidence of their quality and therefore adequate for this work.

Apply the review protocol described early with the results described in Table 2.1

Table 2-1 - SLR Results

Papers on dataset	Count
Papers found	468
Considered after exclusion criteria	398
Effectively used in this work	58

Figure 2-2 represent the distribution of paper screened and selected over the years,

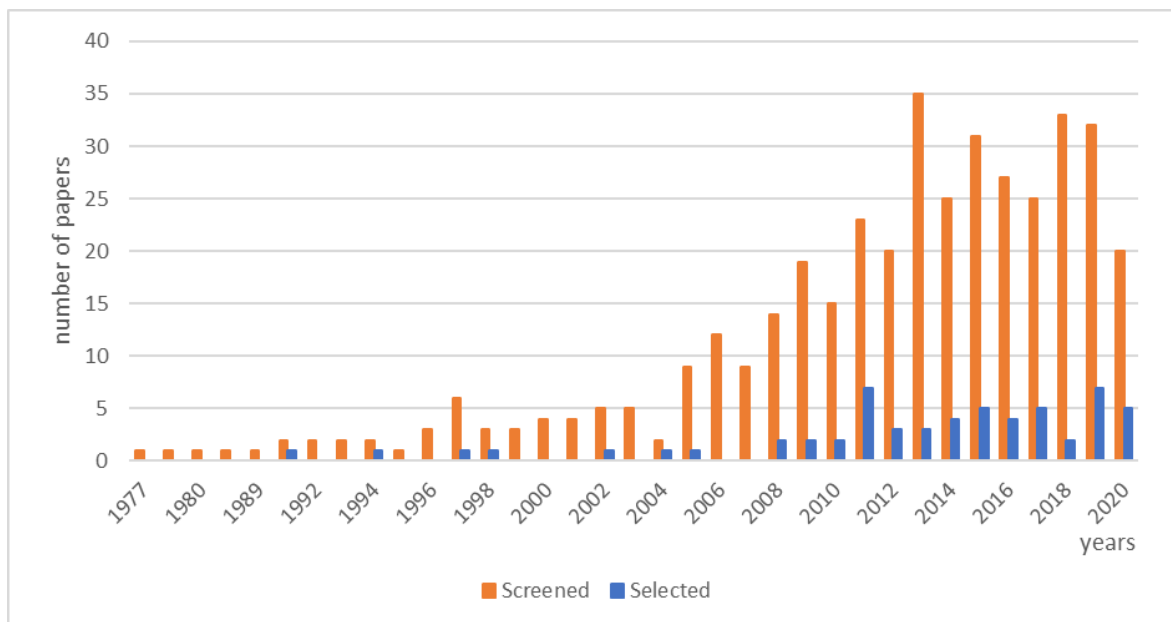


Figure 2-2 - Distribution of the papers screened and selected over the years

Table 2-2 shows that the most common type of publication among selected papers is Journal that reaches almost 80 percent of the set, the other two types represent just 10 percent each.

Table 2-2 - Publication Type of the selected papers

Study	Count	Percentage
Journal paper	46	79,3%
Conference Proceedings and workshops	6	10,3%
Book chapters	6	10,3%

2.1.3 SLR contributes towards answering the research questions.

RQ1. What is the most suitable computational paradigm for this problem?

Parallel computing became a dominant paradigm in the early years of this millennium for High-Performance Computing (HPC) as a reaction to reaching the physical limits of the increased speed of microprocessors. (Dongarra, 2004),(Pacheco, 2011) and also the unstoppable need for more power and computational speed to solve increasingly complex problems, either by the introduction of new algorithms (Agrawal and Srikant, 1994) or by the amount of digital information involved.

Parallel computing means the process of decomposing major problems into smaller ones, where independent parts can be executed simultaneously by several processors and whose results are combined after completion as part of a general algorithm. The main purpose of this paradigm is to increase the computing power available for faster application processing and problem-solving.

In all the studies analyzed it is not always clear whether the approach refers to highly dimensioned datasets (columns) or high amounts of records (rows), or even massive amounts of unstructured data. however, in all analyzed scenarios there is a notable convergence on the solution to follow parallel processing and distributed computing.

By affinity with our theme, we started this analysis with (Zhou *et al.*, 2014) that describes a parallel feature selection method for classification which scales up to very high dimensions and large data sizes, using a method inspired by group testing theory, where the authors conclude that only by parallelizing algorithms is it possible to obtain results in an acceptable time.

Domenico Talia, (Talia, 2002), considered that data mining and knowledge discovery on large amounts of data can benefit from the use of parallel computers both to improve performance and quality of data selection. To achieve this goal, the author presents and describes different forms of parallelism that can be used in data mining techniques and algorithms. Also stipulates three main strategies of parallelism: independent, task, Single Program Multiple Data (SPMD), and hybrid or mixed approaches with the combination of diverse strategies. Also presents possible ways to exploit parallelism, in some data mining techniques, such as “rule induction, clustering algorithms, decision trees, genetic algorithms, and neural networks”. The author concludes that “Knowledge discovery is an area in which parallel computing can be used in a very profitable way”.

According to (Kubica, Singh and Sorokina, 2011), the “problem of feature selection continues to grow in both importance and difficulty as extremely high-dimensional data sets become the standard in real-world machine learning tasks” and describes a Parallel Full Forward Feature Selection approach and a Single Feature Optimization (SFO) heuristic applied to a map-reduce framework among others algorithms.

An adaptable parallel algorithm for direct numerical simulation (DNS) is used to simulate complex laminar and turbulent flow problems on time and space (3D). was demonstrated by (Bolis *et al.*, 2016). This type of

experiment, which can only be performed using parallel processing, requires various parallel strategies and the need to carefully analyze the different possibilities, because some strategies may not perform efficiently, and inefficiency means more time spent, more energy consumed, and therefore more costs.

ShakeMap is a software package used by U.S. Geological Survey (USGS), to estimate median ground shaking in conjunction with observed seismic data. Where the wish for near real-time speeds requires the utilization of parallel processing demonstrated by (Verros *et al.*, 2017).

The need for parallel processing is not a scientific exclusive. (Edosio, 2014) demonstrates the use of Hadoop Architecture and Hadoop distributed storage file system (HDFS) for MapReduce programming model running in a cluster of machines enable organizations with advanced means of analyzing large amounts of data.

The most extreme example is the implementation of scalable data analysis applications at Exascale computing systems such as the ones required to handle data from ATLAS detector at the Large Hadron Collider at CERN, which is a very complex job and it “requires high-level fine-grain parallel models, appropriate programming constructs and skills in parallel and distributed programming”. (Talia, 2019) At this massive scale, the author preconizes distributed memory as a programming model and the use of parallel APIs such as Message Passing Interface (MPI), Bulk Synchronous Parallel (BSP), or Pig Latin a language integrated into Apache Pig, among others.

This author raises yet another interesting topic: currently, commercial “cloud computing platforms and parallel computing systems represent two different technological solutions for addressing the computational and data storage needs of big data mining and parallel knowledge discovery applications”. In addition to the notorious confluence of opinions of all the studies about the absolute need to use parallel computing to address the challenges posed to science today, we can find a surprising trend: the "intrusion" of commercial cloud platforms and the growing adoption of cloud services at the expense of HPC scientific infrastructures. All of the following papers somehow pinpoint in that direction.

An interesting comparison between HPC and commercial (big data) paradigms. Was carried out by (Asaadi, Khaldi and Chapman, 2016). These authors argue that to meet the specialized needs of scientists, there is a need for convergence between HPC and Big Data ecosystems, and they present a data-supported, comparative survey of the main current HPC and Big Data programming interfaces.

In the same line, (Vecchiola, Pandey and Buyya, 2009), presents two practical applications of scientific computing in the Cloud. Both case studies have been implemented on top of the Amazon EC2 infrastructure using Aneka architecture containers. The results of both experiments (classification of gene expression data and Functional magnetic resonance imaging workflows) are compared to the results from a scientific grid and the authors conclude that, after pondering the trade-offs between cost and performance, the public cloud model could be an effective alternative.

Another interesting comparative study between a scientific HPC environment and a private cloud environment built on top of OpenNebula is carried out by (Coutinho, Paillard and De Souza, 2014), in the task of running different scientific experiments, which allowed to determine the differences between the performance, cost, and flexibility of the two environments. In the study, they do not use real data and real problems, which limits

the conclusions, but still highlights the advantage of the characteristic elasticity of the cloud, with satisfactory results regarding the setup of the infrastructure and running the experiments.

Also (Iosup *et al.*, 2011) performed a similar analysis of cloud computing services for many scientific tasks and concluded that they became an alternative for scientists to clusters, grids, and parallel production environments. However, they also indicate that the current commercial offer needs performance improvement to be useful to the scientific community.

Aligned with the same line of reasoning (Saif and Wazir, 2018) presents a survey for performance-based comparative analysis of cloud-based big data frameworks from leading players like such as Microsoft Azure, IBM Cloud, Google Cloud Platform (GCP), or Amazon Web Services (AWS). This is presented from a business point of view rather than a scientific one.

Finally (El-Seoud *et al.*, 2017) agree that enterprises started moving the data and the workloads towards the cloud and parallel processing, following the scientific community, the beginning of the millennium when the "scientific simulations began using massively parallel processing (MPP) systems".

Because of these indications, we can conclude that the computational paradigm to be followed in this work is parallel computing, but it should preferably be carried out in an HPC infrastructure, such as the one of the INCD, although a commercial cloud service can be considered as an alternative.

RQ2. Is the HDF5 format suitable for use in distributed computing and parallel processing environments?

Since the beginning of the 1990s, the HDF (www.hdfgroup.org) and NetCDF (www.unidata.ucar.edu) file formats "have become *de facto* standards for storing, managing, and exchanging data in science and engineering communities" (Pourmal, Cheng and Ayd, 2009). With the introduction of HDF5 (The HDF Group, no date) in 1998, NASA agreed to the unification between the two formats, which happened in 2007 with the release of a new version of netCDF-4 that was built on top of HDF5. In the next year, NASA endorsed HDF5 as a data standard for Earth Science Data.

Currently, bioinformatics applications use HDF5 for storage and management of huge DNA sequencing data, but this is just one example. Several scientific and engineering disciplines such as physics, cosmology, medicine, meteorology, seismology, or behavioral neurobiology rely on HDF5's capacity and robustness, coupled with the existence of open source and commercial tools for analyzing and visualizing data stored in HDF5 format, that together have made HDF5 ecosystem "an attractive standard data format for companies and government organizations concerned with reducing data management costs." (Pourmal, Cheng and Ayd, 2009).

Over the years, the number of applications that use HDF5 successfully in areas outside the scientific and engineering fields has increased, being one of the most notable, the use in the production of visual effects for the "Lord of the Ring" film trilogy.

According to (The HDF Group, no date) HDF5 is a user-defined format like XML and also a binary format allowing compression to help reduce the size of data files. It is also a cross-platform parallel I/O library that is

used by a wide variety of HPC applications for the flexibility of its hierarchical object-database representation of scientific data (Howison *et al.*, 2010). But there are more elements in this equation since the HDF5 technology suite consists of a data model, a library, and a file format for storing and managing data. (Folk *et al.*, 2011) It supports an unlimited variety of data types and is designed for flexible and efficient I/O and high-volume and complex data. HDF5 is portable and is extensible, and also includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format, either directly addressed or through an intermediate layer sitting between the applications and the underlying storage systems (Aaron *et al.*, 2020)(Mehta *et al.*, 2012) Figure 2-3 presents a schematic view of the HDF5 software architecture. According to the HDF5 intro (The HDF Group, no date).

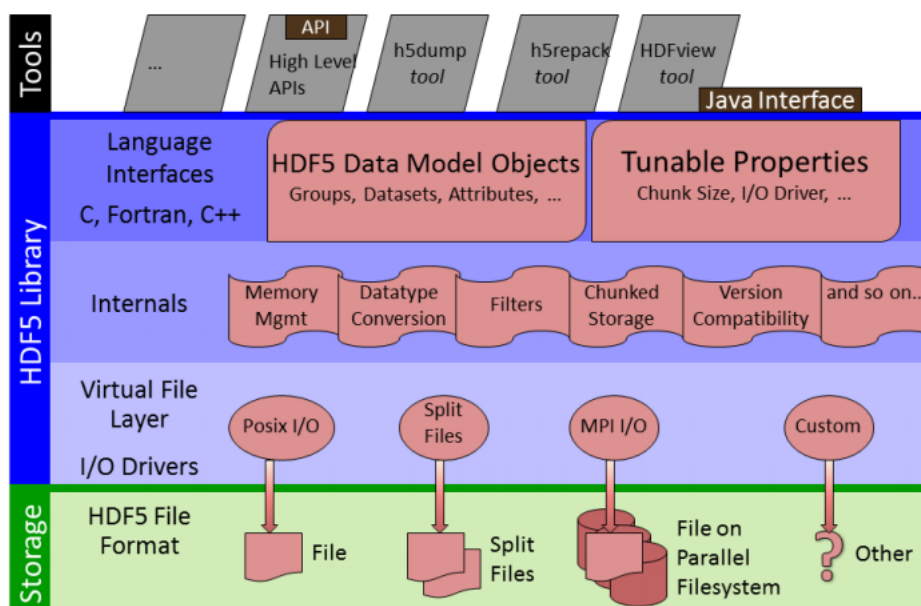


Figure 2-3 - Schematic of the HDF5 software architecture

The list of usage examples is vast and diverse but grouped into two base scenarios: a) porting existing formats to HDF5 or b) creating new formats from scratch. In both these scenarios built on top of HDF5, we have chosen some relevant and exemplary ones that we present below.

The CADISHI software package, which calculates histograms of pair-distances of ensembles of particles with parallel processing on CPUs and GPUs. The calculation of radial distribution functions via distance binning requires the evaluation of $O(N^2)$ particle-pair distances and data input and output are performed efficiently via HDF5. As described by (Reuter and Köfinger, 2019).

Also (Incardona *et al.*, 2019) present OpenFPM as a software library that implements a scalable open framework for particle and particle-mesh codes on parallel computers. It uses parallel and distributed HDF5 files for data archiving.

A high-throughput data compression scheme for astronomical radio data that obtains a very high compression ratio stored on HDF5 was presented by (Masui *et al.*, 2015). Also (Price, Barsdell and Greenhill, 2015) describe how the FITS (Flexible Image Transport System) data format has been the *de facto* data format for astronomy-related data products since 1979 and can be ported to HDF5 in a “straightforward manner” that allowed the faster reading of data (up to 100x of FITS in some use cases), and improved compression (higher compression ratios and higher throughput).

Hence one of the primary barriers to exploratory investigation is simply manipulating the available data, (Hummel, 2016) presents Gadfly, a framework for analyzing particle-based astrophysical simulation data stored in the HDF5 format using Pandas Data Frames (The pandas development team, no date),(McKinney, 2010).

A data model to store datasets from geosciences in the HDF5 format was proposed by (Ritter *et al.*, 2011), “which enables interdisciplinary collaborations and reuse of data processing techniques over different scientific domains and demonstrate how the data repository facilitates the input of new data from sensor networks, field data collection and the massive datasets obtained from remotely sensed Light Detection and Ranging (LIDAR).

Scientific Data Services (SDS) was introduced by (Dong, Byna and Wu, 2013), as a framework for bridging the performance gap between writing and reading scientific data. SDS reorganizes data to match the read patterns of analysis tasks and enables transparent data reads from the reorganized data. To demonstrate it the authors implemented an HDF5 Virtual Object Layer (VOL) plugin to redirect the HDF5 dataset read calls, to the reorganized data and applied it in two use cases: the first, a large-scale Data Analysis of a Vector Particle-in-Cell (VPIC) simulation modeled magnetic reconnection and the second, Mass Spectrometry Imaging Data Analysis.

Related, (Dong, Byna and Wu, 2014) notes that locating critical information in large scientific files is challenging because “existing solutions need significant user involvement in preparing the data, generating indexes, and answering queries”, and introduces the design and implementation of a parallel querying service to answer the queries efficiently. This new service is integrated with SDS mentioned above.

A disruption database (DDB) was presented by (M. Zhang *et al.*, 2020), dedicated to disruption prediction algorithms to predict the occurrence of disruption and try to avoid or mitigate disruption on tokamak devices. The DDB is built on top of HDF5.

Geant4, described by (Barrand *et al.*, 2019), as a platform for “the simulation of the passage of particles through matter” using Monte Carlo methods and new implementation of g4tools/HDF5, that “allows for an effective treatment/representation of statistical data within the Geant4 context, by providing a clean, light and portable engineering solution to this problem.”

Also (Kurth *et al.*, 2015) indicate how the US Department of Energy's (DOE's) Scientific Discovery Through Advanced Computing (SciDAC) program implements HDF5 I/O considered a “first-rate software suite” for HPC parallel I/O, “highly portable and easy to interface with standard scientific libraries”, with the added advantage

of being a non-proprietary format. Also describes its use in a lattice QCD/QFT experiment in the quantum field theory.

Nektar ++ is an open-source framework that provides a flexible, high-performance, and scalable platform for the development of solvers for partial differential equations, which (Moxey *et al.*, 2020) in the new version replaces the existing XML format for the new HDF5 based format significantly faster than the former.

It is noteworthy that not all authors recommend the HDF5 format in their work, some of them highlight potential problems in use, and others even suggest improvements that can be made. For example (Sreepathi *et al.*, 2013) experience indicated that parallel I/O libraries such as HDF5 that rely on Message Passing Interface (MPI-IO), do not scale well beyond 10K processor cores, especially on parallel file systems (like Lustre) and proposed an alternative called SCORPIO.

Also, (Otoo, Nimako and Ohene-Kwofie, 2012), describe an implementation alternative of storage called chunked extendible dense arrays as an approach of storing multi-dimensional dense array on physical storage devices, that allows arbitrary extensions of the array bounds, without reorganizing previously allocated array elements because this extension on HDF5 has a high computational cost.

Finally, (Ertl, Frisch and Mundani, 2017) show considerations and implementation aspects of an alternative I/O kernel based on HDF5 that supports fast checkpointing, restarting, and selective visualization using a single shared output file for an existing computational fluid dynamics (CFD) framework.

So, from the above-mentioned literature review, we can conclude that the widely used HDF5 format is well suited to the purposes of this work.

2.1.4 Related Work

The performed SLR also allowed a sample of related works in this area, which are described below.

A suite of methods and algorithms for model reduction, feature extraction, and classification, especially suitable for large-scale problems. Is proposed by (Phan and Cichocki, 2010). Using Tensors (i.e., multi-way arrays), Tensor decompositions and higher-order (multilinear) discriminant analysis (HODA), and Higher-Order Orthogonal Iterations (HOOI) algorithms, the paper also provide three examples illustrating the validity and performance of the proposed algorithms.

A method inspired by group testing theory was proposed by (Zhou *et al.*, 2014). According to which the FS procedure consists of a battery of random tests processed in parallel. Each of them corresponds to a subset of features, to which a scoring function is applied to measure the relevance of the feature in the classification task.

A Variable Neighbourhood Search (VNS) metaheuristic was proposed by (López *et al.*, 2015). In this approach, the grouping into subsets of features is performed using the Markov blanket concept. To demonstrate it, they conducted experiments on several high-dimensional datasets from two different domains (microarray and text

mining) and compared them with three FS algorithms: Fast Correlation Based Filter (FCBF), A clustering-Based on minimum spanning tree (MST), an algorithm called FAST, and CVNS a similar VNS algorithm.

A One-vs-Rest (OvR)-type extension of the LAD algorithm to multi-class classification problem is presented by (Subasi and Avila-Herrera, 2016). The model is based on the multi-class mixed-integer linear programming (MILP) approach and uses parallel programming to speed up the computations and conclude that is “a very promising option to solve multi-class classification problems”.

A graph-based method for FS was proposed by (Roffo and Melzi, 2017) this method “ranks features by identifying the most important ones in an arbitrary set of cues” by mapping the problem in an affinity graph. In this graph, the features are the nodes, and the solution is obtained by evaluating the importance of nodes through a centrality indicator, specifically, the Centrality Autovector (EC).

Finally, (Brankovic, Hosseini and Piroddi, 2019) present an approach suitable for classification problems with high data dimensionality and complex data distributions, such as DNA microarray datasets. The approach is based on the combination of the following three factors:

- (i) A selection criterion based on the distance correlation (dCor).
- (ii) A distributed combinatorial optimization approach.
- (iii) A randomized FS procedure.

The proposed method has been tested on several microarray datasets, resulting “in quite compact and accurate models obtained at a reasonable computational cost”.

3. Feature Selection with Logical Analysis of Inconsistent Data (LAID)

Data mining is a step in the process of knowledge discovery and can be defined as the process of discovering patterns and extracting knowledge from large amounts of data (Han, Kamber and Pei, 2012). Patterns on data is a generic expression that aggregates concepts like identifying groupings of data not previously known, detection of unusual records, or finding unknown dependencies. This process can be automatic or semi-automatic (Witten *et al.*, 2017). The huge size of many datasets, their distribution as well as the computational complexity required by some data mining methods motivate the development of data-intensive algorithms to parallel and distributed. Technically speaking, these algorithms partition the data into "pieces". Process each piece in parallel, searching for patterns. With or without interaction between the various parallel processes. Finally, the patterns found in each partition are eventually merged to form the final pattern (Han, Kamber and Pei, 2012). As it is often impracticable to analyze the entire Dataset, it is necessary to know how to choose which features the analysis should focus on, to the detriment of others this is the purpose of dimensionality reduction techniques.

3.1 Feature Selection

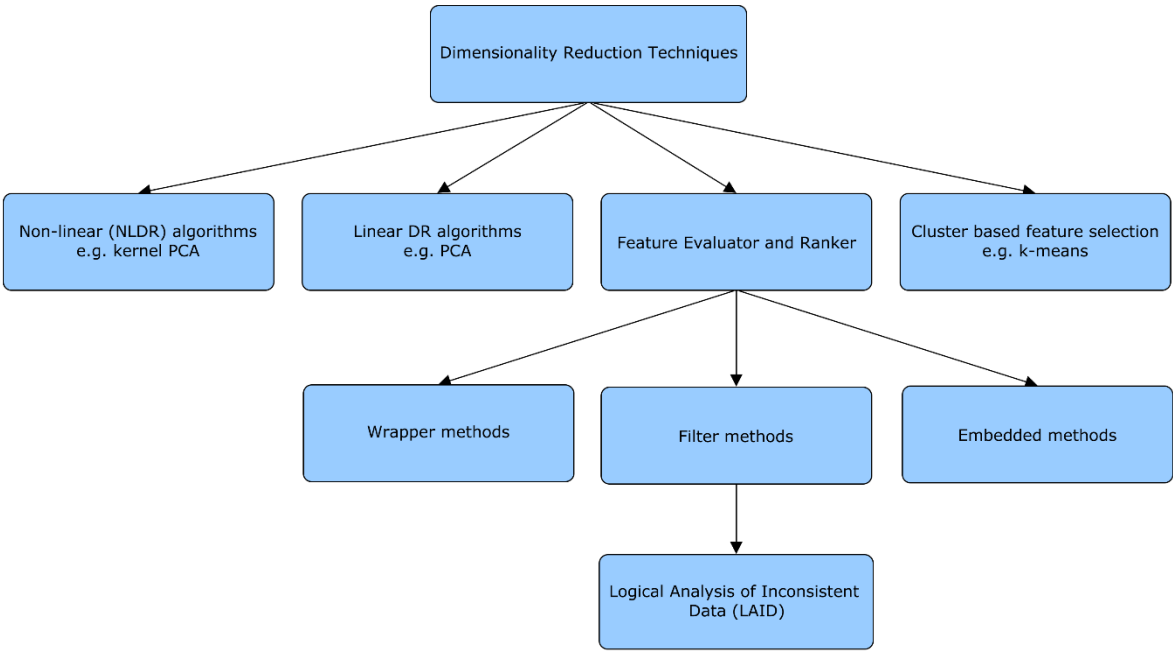


Figure 3-1 - Categorization of dimension reduction methodologies

Figure 3-1 adapted from (Sharma and Saroha, 2015) illustrates the logical analysis of inconsistent data (LAID) a method of feature selection (FS) in the context of dimensionality reduction techniques. FS is a central problem in machine learning. It consists in identifying a representative set of features from which to build a classification model for a specific problem or task. The essential problem with the learning algorithm is to select some subset of features to focus the attention on, given that its variability is representative of the whole set.

Being able to ignore the rest (Kohavi and John, 1998). FS techniques that follow the feature evaluation and ranking approach, are usually cataloged in three major groups: wrappers, which use classifiers to score a given subset of features, this approach tries to identify the best subset of features to use with a specific algorithm; built-in or embedded methods, which inject the selection process itself into the learning of the classifier and finally the filter methods, which analyze the intrinsic properties of the data, excluding the classifier from the process (Roffo and Melzi, 2017).

In this work LAID a Filter approach method is used, the models that follow this approach are divided into two sequential steps. The feature selection step is performed before the prediction model and there is no interaction between the selection and the prediction model, this way the main goal is to evaluate the merits of the features only from the data (Kohavi and John, 1998).

The feature relevance in the context of reducing the dimensionality of a dataset requires some definitions due to the approach used. It should be noted that in the Filter approach the evaluation of the merit of each feature uses its value in the domain and context of the dataset.

“The optimal feature subset is a subset of all relevant features” (Kumar and Minz, 2014). These authors highlight that in the literature, features are usually classified by their relevance with three qualifiers: irrelevant, weakly relevant, and strongly relevant. For (Kohavi and John, 1997) a feature is strongly relevant if it cannot be removed from the subset of features without lowering the subset's evaluation. It will be weakly relevant if at times it contributes to increasing the subset's valuation. Finally, a feature is irrelevant if it is neither strongly nor weakly relevant.

3.2 Feature Selection Algorithms in the genesis of LAID

The presentation of LAID and the theoretical foundation of the theories and methodologies that are in its genesis were profusely and in detail described in previous works (Cavique, Mendes and Funk, 2011), (Cavique *et al.*, 2013), (Cavique *et al.*, 2018), (Apolónia and Cavique, 2019a) and (Apolónia and Cavique, 2019b) for this reason only the main positive and negative characteristics that directly induced the emergence of LAID are listed here.

3.2.1 Rough Sets

Rough set theory was presented in the early 1980s by Zdzislaw Pawlak (Pawlak, 1991) and later it was also proposed as a tool to discover data dependencies and reduce dimensionality in the form of feature selection.

Is limited to discrete data, but allows the existence of inconsistencies (two observations with the same attribute values, but belonging to different classes) because “determine a lower and an upper approximation for each class, rather than try to correct or exclude data inconsistencies” (Cavique *et al.*, 2018). Also allows the existence of non-dichotomized class values.

3.2.2 LAD

In the late 1980s, Peter Hammer and his team unveiled a new data selection and classification methodology, called Logical Data Analysis (LAD) (Cavique *et al.*, 2018). Its theoretical support is discrete mathematics, with emphasis on the theory of Boolean functions (Boros *et al.*, 1997), for that reason LAD only deals with binary variables which implies that for its use with non-binary data it has to be previously binarized.

Also, the LAD methodology does not deal with inconsistencies such as two observations with the same feature values but belonging to different classes. On the contrary, and as a favorable aspect, it is being able to associate a cost to each feature, which allows selecting the features while minimizing the total cost. (Cavique *et al.*, 2013).

3.3 Logical Analysis of Inconsistent Data - LAID

Both LAD and Rough Sets approaches are a subset of filter models, which goal is to reduce the number of dataset attributes using two phases: first a problem transformation and second an optimization through a reduction in the number of attributes (Cavique, Mendes and Funk, 2011). Their main specificity is to keep the semantics of the data by removing only the redundant data.

Logical Analysis of Inconsistent Data (LAID) was proposed in 2011 by (Cavique, Mendes and Funk, 2011) in the scope of a Paremiologic Study and described by the authors as a blend of the best qualities of each of former methodologies, being able to deal with inconsistent data and non-dichotomized classes, characteristics of Rough Sets, (Rissino and Lambert-Torres, 2009) as well as the computational effectiveness of LAD. Also allows integer attributes, with associated costs. (Cavique *et al.*, 2013) which allows for a greater range of selection strategies.

In 2018 this new methodology was used in a feature selection problem involving one dataset with one million attributes (Cavique *et al.*, 2018), solved using partitions of the problem. This experiment uses a computer cluster running at the Portuguese National Infrastructure for Distributed Computing (INCD) for parallel processing. And later (Apolónia and Cavique, 2019a) revisited the same problem again using LAID and a computational approach: based on serial processing and the dataset stored in high-performance data management and storage suite (HDF5) on disk.

In feature selection the starting point is the dataset $D = \{O, XUC\}$, where $O = \{O_1, O_2, \dots, O_n\}$ is a nonempty set of observations (instances or rows), $X = \{X_1, X_2, \dots, X_n\}$ is a nonempty set of features (attributes or columns), and C is the class attribute (Apolónia and Cavique, 2019a).

3.3.1 LAID Steps and Algorithms

3.3.1.1 Removing redundant observations

Redundant observations are observations with the same attribute values and belonging to the same class. Where m is the number of features of the decision system, o_x and o_y are redundant if:

$$\begin{cases} a_i(o_x) = a_i(o_y) \\ c(o_x) = c(o_y) \end{cases} \quad \forall i = 1, \dots, m \quad (3-1)$$

Removing redundancies is a simple task, just eliminate, row by row, the redundant observations.

3.3.1.2 Remove any inconsistency

Inconsistent observations are observations with the same attribute values and belonging to different classes. Where m is the number of features of the decision system, o_x and o_y are inconsistent if:

$$\begin{cases} a_i(o_x) = a_i(o_y) \\ c(o_x) \neq c(o_y) \end{cases} \quad \forall i = 1, \dots, m \quad (3-2)$$

Consider the degree of inconsistency (gr) of an observation o_x , with $x = 1, \dots, n$, as the number of observations of the Universe of dimension n , inconsistent with o_x . This is the formal definition:

$$gr(o_x) = \#\{y = 1, \dots, n: o_y \in U \wedge (a_i(o_x) = a_i(o_y), \forall i = 1, \dots, m) \wedge c(o_x) \neq c(o_y)\}$$

If $gr(o_x) = 0$, it means that observation o_x has no inconsistencies and the degree of inconsistency of the Universe, $gr(U)$ is equal to the greatest of the degrees of the inconsistency of each observation:

$$gr(U) = \max(gr(o_x)), \forall x = 1, \dots, n \quad (3-3)$$

The value of the Universe's degree of inconsistency is always less than or equal to the number of different values in the class.

(Cavique *et al.*, 2013) propose a simple but elegant system for removing inconsistencies in a dataset, by adding new binary attributes, called jnsq, from the French "*je ne sais quoi*", to differentiate inconsistent observations. The number of new binary attributes needed m^* is calculated by the expression:

$$m^* = \lceil \log_2(1 + gr(U)) \rceil \quad (3-4)$$

Meaning that a new jnsq attribute is needed to distinguish two observations with the same attribute values but belonging to different classes. In the case of three or four observations with equal attribute values but different classes, 2 new jnsq attributes are needed (Apolónia and Cavique, 2019a). This new synthetic features remain in the dataset through the next steps, having the same treatment of original features. The set of original and added synthetic attributes is designated by A^* .

3.3.1.3 Generate the Disjoint Matrix

As with the LAD, the LAID methodology uses the determination of the disjoint matrix M . But unlike the previous one, LAID allows classes with an unlimited number of different values (Cavique *et al.*, 2013).

This process consists of, for each observation, comparing the values of each attribute with the values of the following observations if they have a different class value. if the o_x and the o_y are two observations, such that $c(o_x) \neq c(o_y)$, whose comparison corresponds to row i of disjoint matrix M , the elements of matrix $M(d_{i,j}$, with $j = 1, \dots, m + m^*$) according with (Cavique *et al.*, 2013). will be:

$$d_{i,j} = \begin{cases} 0 & \text{if } a_j(o_x) = a_j(o_y) \\ 1 & \text{if } a_j(o_x) \neq a_j(o_y) \end{cases} \quad (3-5)$$

Because inconsistencies were previously removed, each row of matrix M must have at least one non-null (or non-zero) value. The number of rows of matrix M is less than or equal to $n(n - 1)/2$, value that would correspond to the worst possible case *i.e.*, comparing all n observations (Cavique *et al.*, 2013).

3.3.1.4 Returning the reduced set of features.

Obtaining the smallest support set uses a heuristic for the set covering problem proposed by (Chvatal, 1979) whose applicability in this problem is described by (Cavique *et al.*, 2013). It's an iterative process that intends to determine a subset, S , of A^* . In this initially empty subset S , the features chosen in each iteration are placed. In the end, the elements of S correspond to the selection of features that reduce the support set.

The first step is to sum the values of all the rows in each column of the disjoint matrix M , obtaining the vector s , where:

$$s_j = \sum_{i=1}^n d_{i,j}, \text{ with } j = 1, \dots, m + m^* \quad (3-6)$$

The chosen feature, a_e , is the one that best explains the decision system, that is, the one that fulfills the condition:

$$s(a_e) = \max(s_j), \text{ with } j = 1, \dots, m + m^* \quad (3-7)$$

After choosing the feature to enter the solution, a_e , and updating the solution S , the rows of the matrix M that are explained by this attribute, whose value $d_{i,e} = 1$, as well as the column of the attribute itself, are eliminated (Apolónia and Cavique, 2019a).

Then, the process is restarted with a new iteration, until all the lines of the matrix M are eliminated, as we get the features to add to the solution S .

A detailed description with examples can be found in (Apolónia and Cavique, 2019b).

4. Parallel computation and HPC

4.1 Parallel computing

Considering parallel computing architectures, Flynn's taxonomy, presented Figure 4-1 adapted from (Lawrence Livermore National Laboratory, no date), is frequently used to classify computer architectures. It ranks a system according to the number of instruction streams and the number of data streams it can manage concurrently. (Pacheco, 2011).

According to this taxonomy, SISD stands for Single Instruction stream and Single Data stream and matches the classic Von Neumann system architecture.

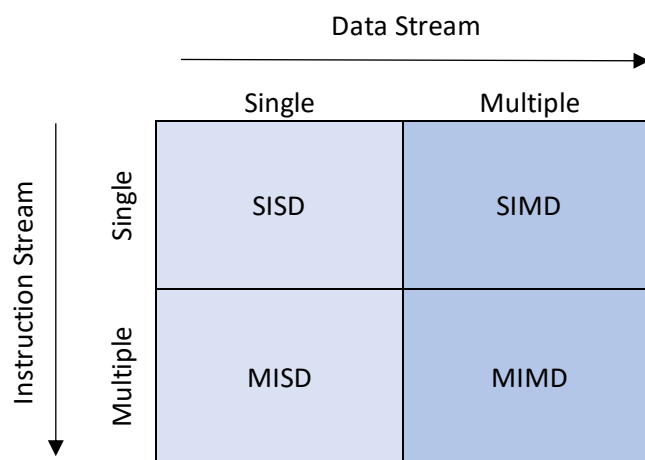


Figure 4-1 - Flynn's taxonomy for computer architectures

MISD systems stand for Multiple instruction Stream, Single Data stream is an uncommon architecture that is generally used for fault tolerance on critical systems. Under this architecture, multiple instructions operate on one data stream.

SIMD refers to a Single instruction Stream, Multiple data stream, this is clearly a parallel system that applies the same instruction to multiple data items. SIMD systems are ideal for parallelizing simple loops that operate on large arrays of data. This form of parallelism is achieved by splitting the data between processors and having all processors apply the same instructions to their data subsets reason that is also called data-parallelism.

MIMD systems stand for Multiple instruction stream, multiple data stream is a system that supports multiple simultaneous instruction streams operating on multiple data streams, unlike SIMD systems, MIMD systems are usually asynchronous. These systems are usually classified by how they share or not memory which implies how the processors can communicate with each other. So, the principal types are shared-memory systems (each processor or core can directly access every memory location) and distributed-memory systems (each processor has its own private memory), on the latter, any communication between processors depends on

being connected over a network and on a message-passing implementation, an example of this systems is a computer cluster or supercomputer.

In this work due to the use of High performance computing a MIMD architecture will be used and described in section 8. Therefore, the techniques used in uniprocessors or shared memory computers are not applicable. Instead, it is assumed that the target architecture is a distributed-memory parallel computer, or in other words, a multicomputer with private memory.

4.2 High-performance Computing - HPC

High-performance computing (HPC), also known as supercomputing, refers to computing systems with extremely high computational power that can solve hugely complex and demanding problems. (Ec.europa.eu, 2021),(Zheng, 2020), performance is commonly measured in floating-point operations per second (FLOPS).

HPC is the focus of this work, but it is relevant to introduce to disambiguate, close computational concepts that are sometimes confused, when essentially, they complement each other, such as Grid Computing, a distributed computing architecture, that utilizes computing resources that are geographically distributed. This architecture provides a virtual environment to users, integrating data and computing resources to accomplish solutions for various types of issues (Cummings and Huskamp, 2005), and also the High Throughput Computing (HTC) paradigm like HPC, HTC's tasks also require huge amounts of computing, but for much longer periods of time. Typically, months or years rather than hours and days.

HPC Infrastructures started as an exclusive to Government and university research facilities, later followed by large companies, but more recently with the advent of commercial Public Cloud, all major Cloud players make HPC available in their service offerings, for example, AWS, Azure batch, IBM HPC, etc.

4.3 Infraestrutura Nacional de Computação Distribuída

INCD - Infraestrutura Nacional de Computação Distribuída is a digital infrastructure supporting research, approved within the framework of the strategic research infrastructures of the FCT - Fundação para a Ciência e a Tecnologia, the Portuguese public agency that supports science, technology, and innovation, in all scientific domains, under the responsibility of the Ministry for Science, Technology and Higher Education and in the context of the Portuguese Roadmap of Research Infrastructures.

INCD provides computing and storage services to the Portuguese scientific and academic community in all areas of knowledge. According to its website the infrastructure is “specially dedicated to providing scientific calculation services, supporting researchers and participation in national and international projects” (FCT – Fundação para a Ciência e a Tecnologia, 2020)

4.3.1 Services provided

- Cloud Computing – that allows run virtual machines with full control over computing resources in an IaaS model based on Openstack.
- HTC Computing - high throughput computing (GRID), Perform thousands of computational tasks to analyze large datasets.
- HPC Computing – high-performance computing, Run applications in parallel processing or in GPUs.

4.3.2 HPC facilities at INCD

The high-performance computing service allows access to batch processing systems equipped with low-latency networks. This service is complemented by a high-performance file system adapted to the needs of parallel processing.

This high-performance computing service enables users to:

- Run parallel processing applications
- Run applications encapsulated in Linux containers
- Accessing GPUs
- Accessing machines with high memory capacity

Three clusters geographically distributed are available.

- INCD-Lisbon (Cirrus-A)
- INCD-Minho (Cirrus-B)
- ISEC-Coimbra (Cirrus-C)

4.4 HPC Environment for the Experiment

INCD kindly provided computational resources in the Cirrus-A HPC cluster, that allowed the execution and testing of the code that constitutes the artifact of this work. These computational resources were made available for this experiment, naturally in concurrence with all other INCD users.

A significant part of the choice of modules and versions used resulted from the excellent support and knowledge of the team that manages this facility.

4.4.1 The Foundations

4.4.1.1 *Cirrus-A cluster on INCD-Lisbon*

Figure 4-2 presents the Technology usage view (Enterprise Architecture notation) of the Cirrus-A cluster and the main components used. On this cluster at INCD-Lisbon, there are 5 batch/compute nodes AMD EPYC 7501 available, running CentOS 7 operative system, with 64 cores per node in a total of 320 cores. Interconnected

by FDR InfiniBand 56Gbps. The INCD filesystems are based on the Lustre distributed filesystem which is mounted in the compute nodes and in the submission/login nodes. The description of the components follows.

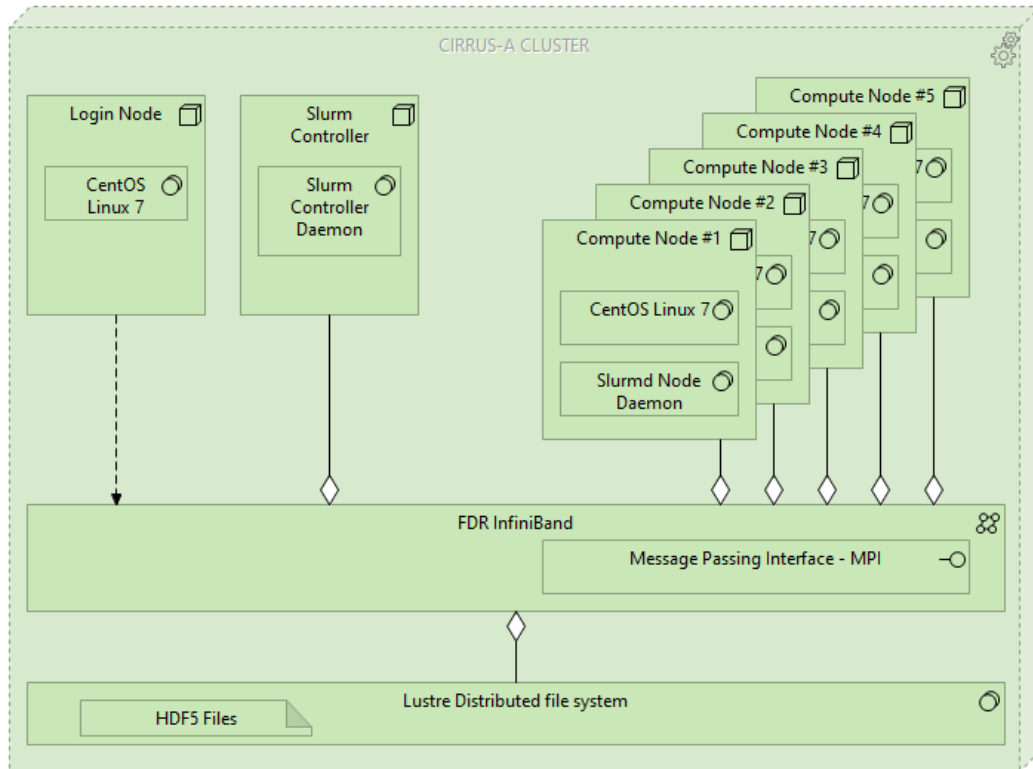


Figure 4-2 - Technology usage of Cirrus-A HPC Cluster and components used

4.4.1.2 CentOS Linux 7 (Core), Operative System

The CentOS is a Linux distribution derived from the sources of Red Hat Enterprise Linux (RHEL). CentOS Linux is at no cost and is free to redistribute. (CentOS Project, no date). Is a common choice for OS in HPC Clusters around the world.

4.4.1.3 Lustre Distributed file system

The Lustre® file system is an open-source, parallel file system widely used for large-scale cluster computing and leadership class HPC simulation environments, with origin in a research project at Carnegie Mellon University, the Lustre file system “has grown into a file system supporting some of the Earth’s most powerful supercomputers.”, (Open Scalable File Systems, no date), like the Department of Energy National Laboratories including Lawrence Livermore, Sandia, Oak Ridge, and Los Alamos’ Cielo supercomputer.

The Lustre file system provides a POSIX compliant file system interface, that can scale to “thousands of clients, petabytes of storage and hundreds of gigabytes per second of I/O bandwidth.” The key components of the

Lustre file system are the Metadata Servers (MDS), the Metadata Targets (MDT), Object Storage Servers (OSS), Object Server Targets (OST), and the Lustre clients. (Open Scalable File Systems, no date). The version used in INCD: 2.12.6

4.4.1.4 Hierarchical Data Format Version 5 (HDF5)

HDF5 is a data model, library, and file format for storing and managing data. “It supports an unlimited variety of data types and is designed for flexible and efficient I/O and high volume and complex data” (The HDF Group, no date).

Since 1987, when the Graphics Foundations Task Force at the US National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign set out to create an architecture-independent software library and file format to address the need to move scientific data among the many different computing platforms in use at NCSA at that time, to our days the Hierarchical Data Format has been adopted across multiple industries and is the de facto standard in the scientific and research community.

Today the HDF Group as a non-profit organization is the developer and supporter of HDF5 and HDF5 related tools, The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format.

The HDF5 version used in this work is version 1.12.0.

4.4.1.5 Slurm Workload manager

Slurm is an open-source, fault-tolerant, and highly scalable cluster management, and job scheduling system for Linux clusters (SchedMD, no date). According to the documentation is made of a slurmd daemon running on each compute node and a central slurmd daemon running on a management node.

As a cluster workload manager, Slurm has three key functions.

1. Allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work.
2. Provides a framework for starting, executing, and monitoring work on the set of allocated nodes.
3. Arbitrates contention for resources by managing a queue of pending work.

Parallel or serial jobs be submitted for execution on compute nodes are scheduled into a queue for execution until enough resources became free for their utilization. All resources are shared with other users and used in concurrency, meaning that every job submitted can impact and be impacted by cluster workload.

Slurm directly launches job tasks and performs initialization of communications through PMIx APIs.

Example of launching jobs: `sbatch mpirun python laid_serial_5.py`

The version used on INCD: 20.02.0.

4.4.1.6 Message-Passing Interface (MPI) standard

MPI (Message-Passing Interface) is a message-passing library interface specification, not an implementation. All parts of this definition are significant. (MPI-Forum, no date) MPI addresses primarily the message-passing parallel programming model, in which data is moved from the address space of one process to that of another process through cooperative operations on each process. Extensions to the “classical” message-passing model are provided in collective operations, remote-memory access operations, dynamic process creation, and parallel I/O (MPI-I/O) that provide routines for file manipulation and data access and allow multiple processes to access a single file.

Point-to-point and collective communication are also supported that allow the coordination of multiple processes like in a master/slave scenario. These communication modes are not used in this work because they require more development and testing time and previous user experience.

Also, in this work, only Independent I/O are used. which allows each process to do I/O independently. Not depending on or being affected by other processes. The exact opposite of Collective I/O.

4.4.1.1 GCC - GNU Compiler Collection

GCC used to stand for the GNU C Compiler, but since this compiler currently supports languages other than C, it now stands for GNU Compiler Collection. All software modules used in this experiment were compiled by INCD technicians using version 4.8 of this compiler. (GNU project and GCC developers, no date)

4.4.2 The building blocks

An application cooperation diagram of the full component stack is presented in Figure 4-3, which also compares a non-parallelization solution using the native HDF5 driver, above and a full-stack with MPI driver that allows parallelization below. This list has been compiled using information from (INCD - Infraestrutura Nacional de Computação Distribuída, no date) and providers’ websites referred in each component, all of them open-source tools.

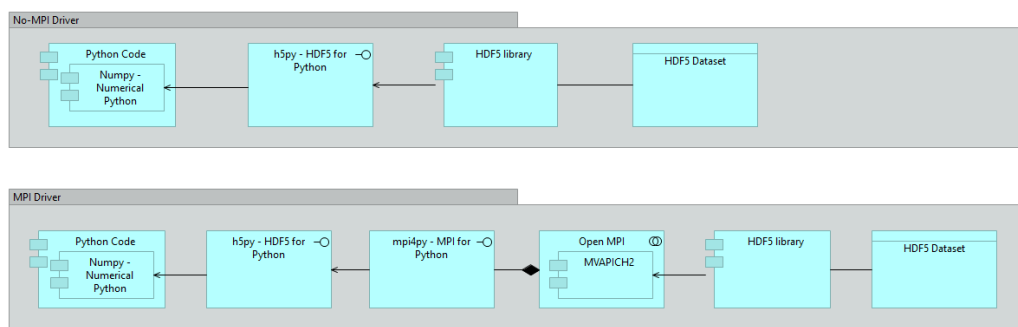


Figure 4-3 - Application cooperation

4.4.2.1 *Open MPI (openmpi)*

Is an open-source High-Performance Message Passing Library for Linux clusters that implements the Message-Passing Interface specification that is developed and maintained by a consortium of academic, research, and industry partners (Open-MPI-Project, no date).

The version used: 4.0.3

4.4.2.2 *mvapich2*

MVAPICH2 is a high-performance MPI-2 implementation for InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE based on MPICH2. This implementation of the MPI standard is developed and maintained by Ohio State University and is available under a BSD licensing (Ohio State University, no date).

The version used: 2.3.5

4.4.2.3 *Python*

Conceived in the late 1980s Python is a widely used general-purpose programming language with increased use in data science. One of its greatest strengths is her large standard library and a huge number of third-party packages that cover a wide range of functionality. It was ported to the most used OS such as Windows, Linux/UNIX, macOS (Python Software Foundation, no date).

Version used 3.6.9

4.4.2.4 *mpi4py - MPI for Python*

MPI for Python package provides Python bindings (interface) for a Message Passing Interface (MPI) implementation, allowing Python applications to exploit multiple processors on workstations, clusters, and supercomputers. MPI for Python provides an object-oriented approach to message passing which grounds on the standard MPI-2 C++ bindings. The interface was designed with a focus on translating MPI syntax and semantics of standard MPI-2 bindings for C++ to Python.

Supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers), with 1-to-many, many-to-1, many-to-many communication of Python objects, as well as efficient communication of Python objects exposing the Python buffer interface (e.g., NumPy arrays). Provides optimized communication with NumPy arrays (Dalcín, Paz and Storti, 2005).

The version used: 3.1

4.4.2.5 h5py - HDF5 for Python

h5py is a high-level Python library for HDF5, such as MATLAB, and IDL. makes available for Python a High-level API which offers the main features of HDF5 in an interface modeled on dictionaries and NumPy arrays also provides a low-level API, which more closely follows the HDF5 C API” (Andrew Collette and contributors, no date)

The version used: 3.1.0

4.4.2.6 NumPy - Numerical Python

NumPy (Numerical Python) is an open-source Python library that’s used in almost every field of science and engineering. It’s the facto standard for working with numerical data in Python, and it’s at the “core of the scientific Python and PyData ecosystems”, (NumPy Developers, no date) also the NumPy API is used extensively in the most used data science and scientific Python packages like Pandas, SciPy, Matplotlib, scikit-learn.

NumPy can be used to perform a “wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices, and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices” (NumPy Developers, no date). The h5py library also extensively uses NumPy, in handling datasets, so the link between both libraries and their use in this work is obvious.

Comparing data types and data structures available in Python and NumPy, reveals greater flexibility in Python, perhaps one of the main reasons that this language became so popular, but flexibility, like (VanderPlas, 2017) remember, “comes at a cost: to allow these flexible types, each item in the list must contain its own type info, reference count, and other information—that is, each item is a complete Python object. In the special case that all variables are of the same type, much of this information is redundant: it can be much more efficient to store data in a fixed-type array”. The difference between a dynamic-type list and a fixed-type like NumPy array is illustrated in Figure 4-4.

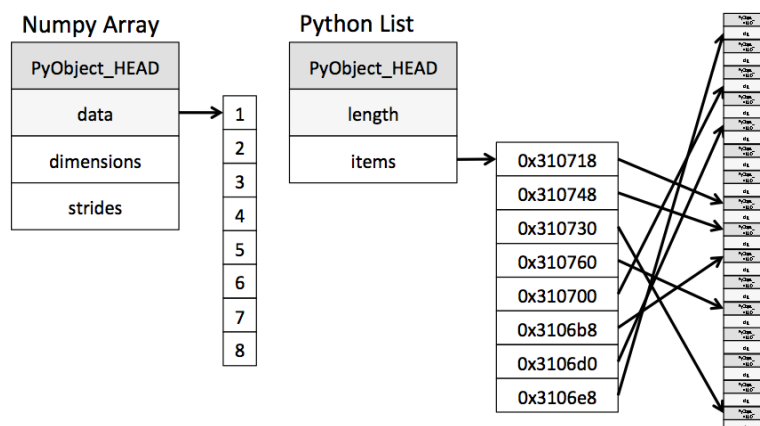


Figure 4-4 - Difference between NumPy Array and Python List

At the implementation level, the array essentially contains a single pointer to one contiguous block of data. The Python list, inversely, contains a pointer to a block of pointers, each of which in turn points to a full Python object. The advantage of the list is flexibility: because each list element is a full structure containing both data and type information, the list can be filled with data of any desired type. Fixed-type NumPy-style arrays lack this flexibility but are much more efficient for storing and manipulating data (VanderPlas, 2017)

The version used: 1.19.5

4.5 Discussion

Using the computing resources of an HPC infrastructure is an opportunity and a challenging task. Setting up the HPC environment to run parallel processing is no task for beginners. It is necessary to be prepared for this and to foresee the time needed to prepare the environment by carefully studying the alternatives of the components, especially modules and libraries needed. To avoid incompatibilities.

It is particularly important to consult the experts on the infrastructure team and follow their advice and guidance, which can save you many hours and frustration.

Given that it is a shared and competitive environment, it is also a good idea to know and follow the facility's protocols and rules, to avoid disturbing the work of other users and collaborate in the more efficient use of the shared resources.

It is also a good idea and a good practice to have a home lab with an environment as similar as possible to the HPC environment, which helps to develop and test code, for example, with a reduced dataset, and thus achieve the maximum amount of development without the need for HPC resources, thus avoiding executions interrupted after hours due to simple avoidable errors in the code.

Regarding the job scheduling and the resources needed, it is advisable to be cautious, asking for too many resources at once when many other cluster users are waiting in the queue it just takes a long time to wait until those resources are available. It is often preferable to request fewer resources and be able to perform jobs faster.

5. Parallel Code

Given the prior availability of a codebase for the LAID algorithms written in the C and Python programming languages, these source codes were used here with emphasis on the Python version that already used HDF5, through in a serial programming model reason for choosing this language and because, thus, part of the work consisted in the adaptation of the sequential code base and the necessary additional developments.

5.1 Parallelism in Python

There are several general methodologies by which Python parallelism can be achieved a non-exhaustive list follows using python standard features or 3rd party libraries:

- Multiple invocations of the Python interpreter
- Process-level and subprocess parallelism
- Thread-level parallelism
- Numba a JIT compiler offers a range of options for parallelizing Python code
- OpenMP API that supports multi-platform shared-memory multiprocessing programming
- Libraries such as BLAS or Intel oneAPI Math Kernel Library
- Higher-level frameworks, like Dask, Joblib, or Ray

Also, Python's standard library provides a multiprocessing package that supports the spawning of processes. This can be used to achieve some level of parallelism within a single compute node. (National Energy Research Scientific Computing Center, no date) “It cannot be used to achieve parallelism across compute nodes”.

As seen in section 0 there is a consensus in the case of using distributed memory as a programming model to resort to parallel APIs, such as Message Passing Interface (MPI) (Talia, 2019), and this solution is available for Python through a third-party library called MPI for Python or more usual mpi4py discussed in more detail below.

5.2 Design Methodology for parallel program

Foster’s methodology, described by (Pacheco, 2011) provides a sequence of steps that can be used to design parallel programs or adapt serial code. The steps are:

- Partitioning - The process of dividing data and computation into small pieces,
- Communication - The process of defining what and how tasks will communicate with each other,
- Agglomeration or aggregation – The process of grouping tasks into larger tasks to simplify and improve performance,
- Mapping - The process to assign tasks to processes/threads/physical processors.

The purpose of partitioning is to discover as much parallelism as possible, which was widely used as described in 5.3. on the contrary, communication was not used at all. Agglomeration was also used on the final run described in section 8.5.

the final step of this methodology is the procedure of assigning each task to a processor, mapping. This technique was intensively used in this work, considering that the objective in the development of mapping algorithms is to minimize the total execution time, which can be obtained by minimizing the communication

between processors (or eliminating it at all) and maximizing the use of each processor. Was also associated with rows or columns data partition

5.3 Coding the LAID algorithms

All the Python code is based or adapted from previous work (Apolónia and Cavique, 2019a) and (Cavique *et al.*, 2018) the changes consisted essentially of code improvements, changes in the execution flow, the introduction of a configuration file, different use of files, and HDFS datasets and introduction of techniques aimed at parallelization. Can be accessed in a public repository referred to in Annex 11.2.2.

Of the various decisions taken, stands out the access of HDF5 files using the MPIIO driver, we opted for using MPI without communication what is sometimes called MPI independent mode. In which each of the tasks works independently without collective or point-to-point communication between nodes or a node master. This decision was taken due to the time available to complete this work as well as the intent to keep a common code base to serial and parallel versions.

5.3.1 Laid steps

According to (Cavique *et al.*, 2018)(Apolónia and Cavique, 2019a), the Logical Analysis of Inconsistent Data algorithm consists of the following steps:

Input: dataset $D = \{O, XUC\}$ with binary variables

Output: (number of features, accuracy)

1. check data inconsistencies and add dummy variable “jnsq” as a discriminant feature to remove any inconsistency in addition check and remove all redundant observations.
2. disjoint matrix generation $[A_{i,j}]$ this is a $O(n^2)$ in the Big O notation
3. number of features = Minimum Set Covering Problem. In this third step of LAID, a heuristic (Chvatal, 1979) is applied to solve the set covering problem. This is an $O(\log n)$ on Big O notation and is known to be NP-hard.
4. accuracy = Cross-validation

5.3.2 Step 1 - Check and fix redundant or inconsistent observations

Redundant and Inconsistent observations, described in 3.3.1.1 and 3.3.1.2 must be handled on this step. Both algorithms with high dependency between their interactions and are inherently considered serial algorithms

Considering that the serial version is fast enough, and no redundancies or inconsistencies were found in the Source-Dataset (1000k columns) or in subsets (200k columns) the option was to keep the existing code with only minor adaptations and give priority to the next steps.

5.3.3 Step 2 - Disjoint Matrix Generation

As described in 3.3.1.3, this step consists of reading the source dataset's rows and comparing them with all those that have a different class value, the disjoint lines found are written in a destination dataset, which for operational reasons has been decided to be stored in a separate HDF5 file, this new derived dataset has been named DM-Dataset.

Because there are no dependencies between each iteration of the algorithm used, i.e., each observation/row of the Source-Dataset is treated independently, its parallelization appears to be simply requiring only a few adaptations to the serial code. What in the parallel jargon is known as an “embarrassingly parallel problem”. Technically it is only necessary to distribute the rows of the source-Dataset by the processors assigned to the run, which can be seen as a way of data-parallelism from the SIMD definition, or a horizontal by row partition.

However, it was not so simple. All parallel tasks on the run write the generated rows to the same target HDF5 dataset. If this recording could be sequential, something like adding new rows to the dataset, the process would be simple, but it can't, the HDF5 datasets used are all fixed length. This implies, to avoid overwriting, strictly controlling, and segregation of the write-row index, as is done for reading.

It is obviously possible to define a formula to calculate rows index, but only if the Source-Dataset does not have redundant rows and if it is sorted by the value of the class(es) and this effectively is. But this sort, despite good practice, is not a requirement of LAID and it cannot be considered an assumption. Which makes this an unsolvable problem, because before the execution of the algorithm it is not possible to know how many, and in which index the rows should be written. This insolubility inspired the solution, not elegant but effective, the algorithm will run twice. In the first pass, it identifies the source and destination rows, without spending time writing to disk. With this valuable information, the second pass knows exactly what and where to write. The bet is that parallelization allows not only to recover the time spent on the first pass but also to save more time because it avoids unnecessary comparisons.

The first trial had a disappointing result. The chart in Figure 5-1, shows the Parallel version with 5 parallel tasks that took a few minutes longer than the serial version. Row-by-row the algorithm wasted time for the serial version.

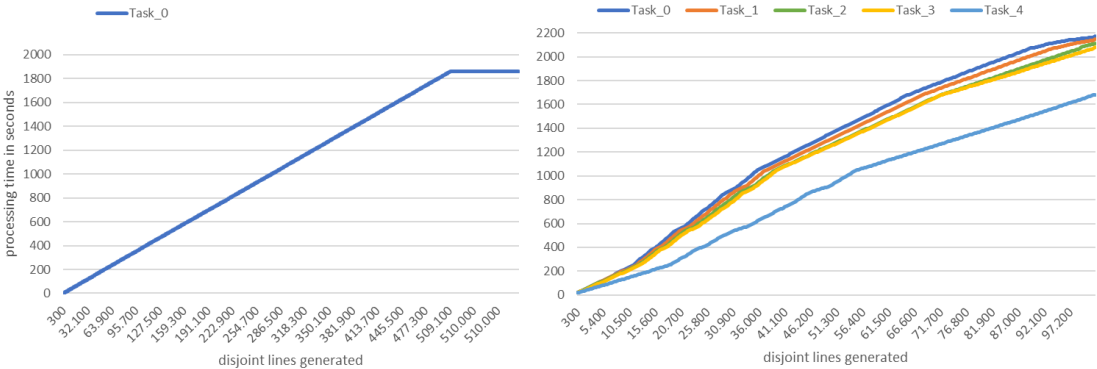


Figure 5-1 - DM-Dataset generation, first trial of serial version against 5 parallel tasks

From the analysis of these results, it was concluded, that whether in parallel or serial version, The DM rows were saved to HDF5 on disk, Row-by-row, immediately after being generated, which caused overhead.

To avoid that issue the code was changed to include a buffer, which allowed the number of write operations to be reduced by writing more data at a time.

The introduction of this buffer has proved so useful that even the serial version benefits from it, (29% from the initial time), the results after buffer introduction are presented in Table 8-2 and the chart is presented in Figure 8-3 of section 8.4.

The conclusion is that it is not enough to parallelize, because the effects of any inefficiencies in the code are equally multiplied, it is necessary to correct these inefficiencies beforehand to take advantage of the parallelization. Listing 5-1 below presents the code of this second pass with the buffer.

```
step = int(class_ref/worker_nodes) # number_of_rows of class ref (1700 on the sample)
start = task_id * step
stop = int( task_id * step + step -1 ) # number_of_rows-1
if task_id + 1 == worker_nodes and stop < number_of_rows: # ensure odd cases last // task
    stop = number_of_rows-1
sys.stdout.write("Task_{}; start; {} stop; {}\n".format(task_id, start, stop))
sys.stdout.write("Task_{}; Build Disjoint Matrix parallel version\n".format(task_id))
disjoint_array = np.zeros((number_of_features + number_jnsq_features),
dtype=np.dtype(np.int8)) # stores compare of mi e mj sums
t1 = time.perf_counter()
rows_on_disjoint_matrix = dset.attrs['rows_on_disjoint_matrix'] # from first pass
size_a = rows_on_disjoint_matrix
size_b = number_of_features + number_jnsq_features
hdf5_disjoint_file = os.path.join(mydir, config_base[0][10]) + '.h5' # one dm file for all tasks
hfdm = h5py.File(hdf5_disjoint_file, 'w', driver='mpio', comm=MPI.COMM_WORLD) # Create, truncate
dataset_disjoint_matrix = hfdm.create_dataset('dmatrix', (size_a, size_b), dtype=
np.dtype(np.int8))
number_of_interact = 0

for i in range(start, stop+1):
    rows_on_disjoint_matrix = dm_guide_array[i,0]
    show_rows_on_disjoint_matrix = rows_on_disjoint_matrix
    comparations_expected = dm_guide_array[i,1]
    number_of_interact_per_row = 0
    rows_on_disjoint_matrix_per_row = 0
    buffer_array = np.zeros((comparations_expected,number_of_features + number_jnsq_features),
dtype=np.dtype(np.int8))
    t1_row = time.perf_counter()

    if comparations_expected > 0: # redundant observations are not considered.
        read_mi_array = dset[i,:number_of_features+number_jnsq_features]
```

```

for j in range(i+1, number_of_rows): # compares current row with following rows
    number_of_interact_per_row += 1
    if (j not in redundant_array) and (class_array[i] != class_array[j]):
        read_mj_array = dset[j,:number_of_features + number_jnsq_features]
        disjoint_array = np.absolute(np.subtract( read_mi_array , read_mj_array ))
        buffer_array[rows_on_disjoint_matrix_per_row,:] = disjoint_array # updates buffer
        rows_on_disjoint_matrix += 1
        rows_on_disjoint_matrix_per_row += 1

    number_of_interact += 1
# updates DM dataset from buffer
dataset_disjoint_matrix[dm_guide_array[i,0]:comparations_expected,:] = buffer_array

t2_row = time.perf_counter()
test = (f"in; {t2_row - t1_row:0.4f}; seconds")
sys.stdout.write("Task_{}; Row;{}; Class;{}; Start DM row;{}; comparations_expected; {};
Number of interact;{}; Disjoint rows found;{}; {}\\n".format(task_id, i, class_array[i],
show_rows_on_disjoint_matrix, comparations_expected,number_of_interact_per_row,
rows_on_disjoint_matrix_per_row, test ))

```

Listing 5-1 - Build Disjoint Matrix - Parallel version

5.3.4 Step 3 - Find the solution with the minimum set covering algorithm

This is the real bottleneck in all steps of the LAID methodology. The serial version used in this step is inherently a serial algorithm. there is a clear dependence between interactions given that the computation performed in the (i + 1)-th iteration depends on the result of the i-th iteration.

The approach followed was to perform the vertical partition of the DM-Dataset columns and their distribution by parallel tasks. But the different solutions found must be compared and the best one chosen. which requires communication and cooperation between processors/parallel tasks, so In this work, the approach for a parallel solution was limited to the first iteration. For this reason, the goal of parallelizing this step 3 was postponed.

Literature reveals various other potential parallel solutions (Blelloch, Simhadri and Tangwongsan, 2012),(Chakravarty and Shekhawat, 1992). there is a common point in all of them relying on some form of communication between processors.

In conclusion, the next approach will have to make use of MPI point-to-point or collective communication modes via shared-memory or external memory that allows coordination between parallel tasks/processors.

5.3.5 Flow control flow and orchestration

By opting for the code aggregation of all blocks in a single python program, some flexibility was lost if one intended to run only one of the steps, the introduction of on/off conditions and their control from the

configuration file, allowed to keep a single block of code, which can be submitted to the cluster for execution at once, keeping control of what is intended to be executed.

5.4 Performance Metrics for Parallel Systems

Execution (run) time: Parallel execution time is usually defined as the time that elapses from the moment a parallel calculation starts until the moment the last processor finishes execution.

Whereas the main objective of parallel computing is increasing speed and reducing time complexity is necessary to measure the result. for this purpose, two performance metrics are usually used (Rastogi and Zaheer, 2016), (Cavique *et al.*, 2018) to measure the efficiency of an algorithm in terms of the time complexity factor. N computers working simultaneously can increase the speed up to N times, or an approximation of N as other factors can influence the time spent, such as bottlenecks communications.

The Speedup S_p factor is defined as the ratio of the serial runtime of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors.

Can be calculated as:

$$S_p = \frac{\text{Execution time using one processor (best sequential algorithm)}}{\text{Execution time using a multiprocessor with } p \text{ processors}} = \frac{t_s}{t_p} \quad (5-1)$$

Where t_s (T_{serial}) is the execution time on a single processor or serial run time.

t_p (T_{parallel}) is the execution time on a multiprocessor or parallel run time.

S_p it the increase in speed by using multiprocessor.

To determine t_s and t_p , it is necessary to include calls to a timer function in the source code.

Was also calculated the Efficiency E is defined as the ratio of speedup S_p to the number of processors P . Efficiency measures the fraction of time for which a processor is usefully utilized.

$$E = \frac{\text{Speedup } S_p}{\text{number of processors } P} \quad (5-2)$$

6. The Dataset used

The dataset used on this work was also used in the 2018 experiment described by (Cavique *et al.*, 2018). Contains a total of 2 000 observations (by row) x 1 000 000 features (by column) of synthetic data representing some kind of Omic data. All observations are labeled according to a binary classification.

It is subdivided into ten text files containing a sequence of pairs of digits that represent coordinates (row and column or in this context observation and feature) of cells with a value of "1", as presented in Figure 6-1. Half of the files contain the observations labeled as a class "0". The other half contains class "1" observations.

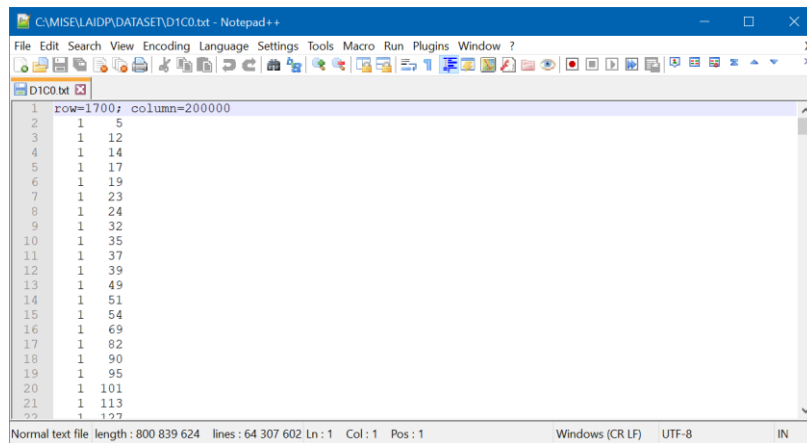


Figure 6-1 - Fragment of one of the text files

This way every one of the class "0" subset represents 1 700 x 200 000 cells coordinates of the original matrix and the class "1" represents 300 x 200 000 cells coordinates. As described in Table 6-1.

Table 6-1 - Rows, Columns, Classification, and size on disk of Dataset parts

File Part	Class	Part	Rows	Columns	Nonzero cells	Size in Mb
D1C0.TXT	0	1	1 700	200 000	64 307 600	764
D1C1.TXT	1	1	300	200 000	13 783 200	164
D2C0.TXT	0	2	1 700	200 000	64 681 600	768
D2C1.TXT	1	2	300	200 000	13 925 700	165
D3C0.TXT	0	3	1 700	200 000	64 591 500	767
D3C1.TXT	1	3	300	200 000	13 665 300	162
D4C0.TXT	0	4	1 700	200 000	64 117 200	762
D4C1.TXT	1	4	300	200 000	13 806 900	770
D5C0.TXT	0	5	1 700	200 000	64 795 500	770
D5C1.TXT	1	5	300	200 000	13 768 500	163
					391 443 000	5 254

Table 6-2 - Time for loading all text files, performed by two parallel tasks

Task	order on batch	batch	minutes	filled cells	Cells filled per minute
Task_0	1	D1C0	17,74	64 307 600	3 625 309,4
Task_0	2	D1C1	4,83	13 783 200	2 852 866,2
Task_0	3	D2C0	20,88	64 681 600	3 097 206,2
Task_0	4	D2C1	5,27	13 925 700	2 642 048,4
Task_0	5	D3C0	17,88	64 591 500	3 612 387,5
Task_0 Totals			66,61	221 289 600	3 322 413,7
Task_1	1	D3C1	3,83	13 665 300	3 565 929,1
Task_1	2	D4C0	25,60	64 117 200	2 504 834,6
Task_1	3	D4C1	5,44	13 806 900	2 538 886,4
Task_1	4	D5C0	18,12	64 795 500	3 576 628,4
Task_1	5	D5C1	3,84	13 768 500	3 582 491,5
Task_1 Totals			56,83	170 153 400	2 994 215,5

For this case, 10 text files are equivalent to 10 work packages. The entire loading process is regulated in a configuration file named config.json which is also part of the artifact (See Annex 11.2.1) this config file contains the original coordinates of the entire block on the Source-Dataset and its translation to HDF5 coordinates, each block is described individually as shown on Figure 6-3.

```
[
  [ "D1C0", "D1C0.txt", 0, 0, 0, 1699, 199999, 0 ],
  [ "D1C1", "D1C1.txt", 1, 1700, 0, 1999, 199999, 0 ],
  [ "D2C0", "D2C0.txt", 0, 0, 200000, 1699, 399999, 0 ],
  [ "D2C1", "D2C1.txt", 1, 1700, 200000, 1999, 399999, 0 ],
  [ "D3C0", "D3C0.txt", 0, 0, 400000, 1699, 599999, 0 ],
  [ "D3C1", "D3C1.txt", 1, 1700, 400000, 1999, 599999, 0 ],
  [ "D4C0", "D4C0.txt", 0, 0, 600000, 1699, 799999, 0 ],
  [ "D4C1", "D4C1.txt", 1, 1700, 600000, 1999, 799999, 0 ],
  [ "D5C0", "D5C0.txt", 0, 0, 800000, 1699, 999999, 1 ],
  [ "D5C1", "D5C1.txt", 1, 1700, 800000, 1999, 999999, 1 ]
],
```

Figure 6-3 - Work packages described on the config file

For 10 work packages, the most suitable number of parallel tasks are 1 (serial load), 2, 5, or 10, which ensure the best load distributions by work packages. Adding more than 10 tasks does nothing to reduce time.

The final version of the code added to the artifact does not have separate versions for parallel or serial processing, It's the Rank (MPI Job Task Id) that defines this behavior, if equal to 1 (single task), then processing follows Serial Programming model, if greater than 1, it is Parallel programming model.

This is an example of application the Mapping referred on Foster's methodology. This allows an existing function in the code to map the work to be done by the different parallel tasks in execution demonstrated in Listing 6-1.

```

def map_parallel_worker_to_workpack(rank, worker_nodes, work_packages ):
    step = work_packages / worker_nodes
    ini = int(rank * step)
    end = int(ini + step -1)
    return (ini, end)

```

Listing 6-1 - Map Function

As this code was never executed by just one task, it is only possible to have a duration estimate that points to approximately 123,44 minutes (sum of the minutes spent by the two tasks in Table 6-2). In the same way, we estimate that the minimum time required if using a maximum of 10 parallel tasks (one for each text file) is equivalent to the longest of the batches on the run described in Table 6-2 bold cell i.e., 25,6 minutes which implies a Speedup factor of 4.82 with an Efficiency of 0.48 as shown on Table 6-3.

Table 6-3 - Data loading into HDF5, Speedup, and Efficiency Estimated

T _{serial}	T _{parallel}	Processors	Speedup Sp	Efficiency E
123,44	25,60	10	4,82	0,48

6.3 A brief exploratory data analysis

After loading the data to HDF5 and to better understand the characteristics of the dataset, a brief exploratory data analysis was carried out allowing us to conclude that the total number of cells in the dataset is 2 000 000 000 (2 000 x 1 000 000) whereas only 391 443 000, (19.57%), are filled with non-zero value means a sparse matrix. This sparsity is not uniform across the columns, in fact, 311 704 (31%) of the columns don't contain any value as shown in Table 6-4, and only 19% of the columns concentrate 50% of the total cells filled.

Table 6-4 - Sum of values per column vs number of columns

#	Sum of value per column	Number of columns
1	0	311 704
2	300	92 980
3	1 700	73 155
4	2 000	22 162
5	390	11 248
6	393	11 213
7	391	11 171
(...)	(...)	(...)
158	314	1
159	310	1

The chart in Figure 6-4 graphically confirms that the distribution of the sum of column values follows a long tail.

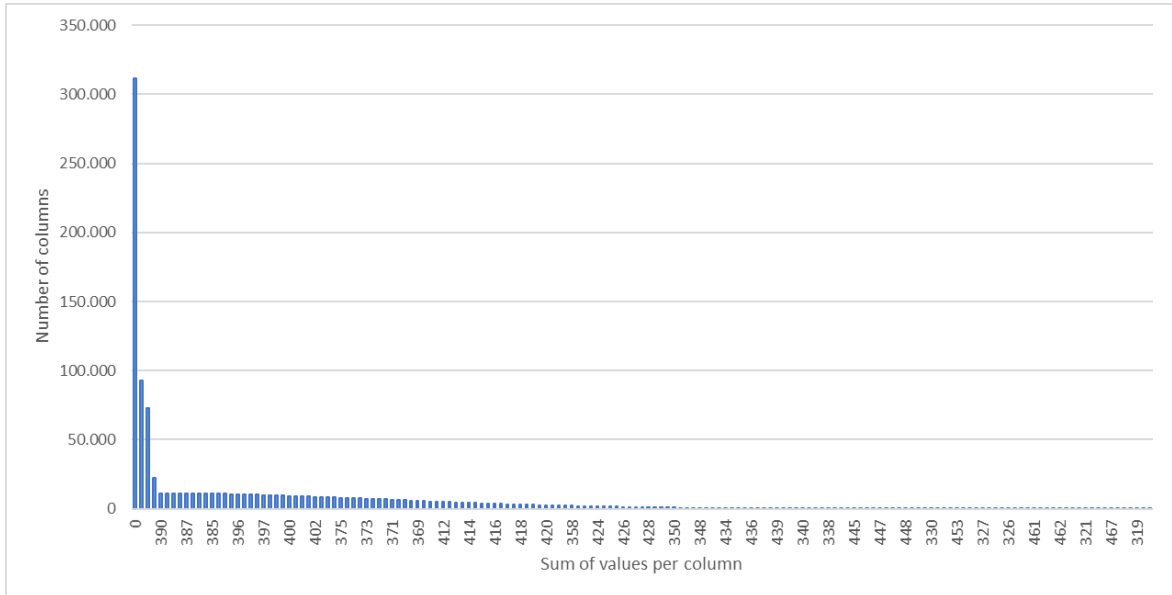


Figure 6-4 - Distribution of the sum of values in column

It is also relevant to note that the pattern of distribution is not uniform, on column ranges [1 to 100k], [200k to 300k], [400k to 500k], [600k to 700k] and [800k to 900k], (i.e the first 100k of each text file) a random distribution pattern predominates, visible on Figure 6-5 pattern a), while the remaining presents a more organized and concentrated distribution Figure 6-5 pattern b), where a small number of columns contain most values and there are a large number of columns without any value.

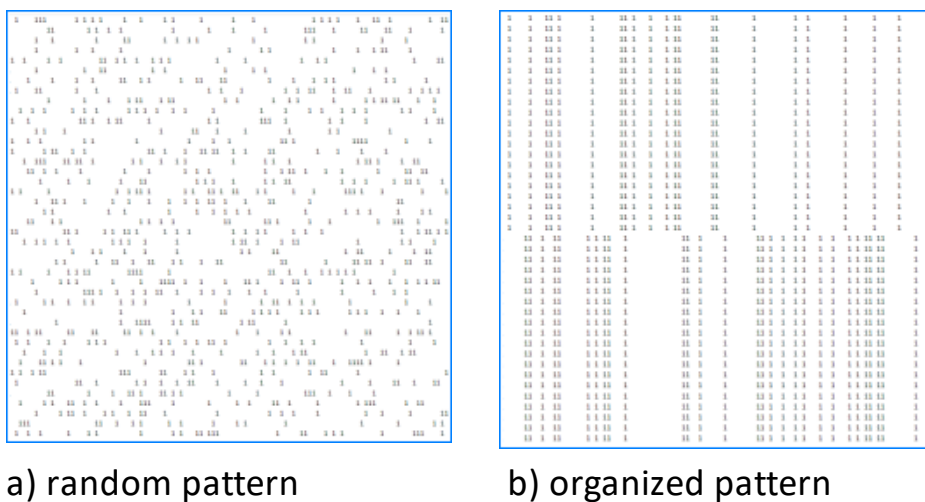


Figure 6-5 - Distributions patterns found on Source-Dataset

6.4 Discussion

Loading the dataset is also an excellent exercise for testing parallel computing techniques, which substantially reduced loading time.

The main conclusion is to avoid the cell-by-cell storage method as it is too slow. It is preferable to use h5py/HDF5 capability and store 2d arrays with contiguous data blocks at once.

After loading it is also important to validate the information stored on the HDF5 dataset,

1. Counting the number of non-zero cells (sum) should be the same reported on the column “Number nonzero cells” in Table 6-1, to ensure all data was stored on the target dataset.
2. Check non-zero on known cells on the boundaries of different File Parts, to ensure the correct placement of all parts.
3. Check all the values on the labeled column, to ensure that the correct class value was correctly loaded and stored.

It is also advisable and desirable to know as much as possible about the characteristics of the dataset to be processed. Considering its large dimensions this is by itself a challenging task where visual inspection is not even possible and descriptive statistics do not help.

The ranges of columns where this pattern b) predominates if processed separately (via vertical partition per column), has implications for the outcome due to the high number of redundant observations. Which naturally limits the number of possible partitions and of course the number of parallel tasks if parallelization via vertical partition is used.

7.HDF5 Storage Layout: Chunking Analyses

7.1 Storage layout options

The storage layout or storage strategy must be aligned with dataset access patterns which mean read operations, hence the choice to test the storage layout options with the block of code that generates the DM-Dataset, in this step the access to the Source-Dataset is done exclusively by reading an entire line, never by column.

As mentioned in (Andrew Collette and contributors, no date) “The default storage layout of HDF5 files is contiguous storage: data of a multidimensional array is serialized (or flattened) along the fastest-changing dimension and is stored as a contiguous block in the file. This storage mechanism is recommended if the size of a dataset is known and the storage size for the dataset is acceptable to the user”. in other words, laid out on disk in traditional C order.

Datasets may also be created using HDF5’s chunked storage layout. This means the dataset is divided up into regularly sized pieces which are stored haphazardly on disk and indexed using a B-tree. This storage makes also it possible to resize datasets, and because the data is stored in fixed-size chunks, to use compression filters.

Chunking has performance implications (Andrew Collette and contributors, no date). It’s recommended to keep the total size of chunks between 10 KiB and 1 MiB, larger for larger datasets. And more relevant when any element in a chunk is accessed, the entire chunk is read from the disk. If only use a subset of the data, the extra time spent reading from the disk is wasted. it is necessary to be aware that chunks bigger than 1 MiB by default will not participate in the fast, in-memory “chunk cache” and will instead be read from the disk every time.

About the chunk size, they can't be too small and can't be too big either and there's a limit. The current maximum number of bytes in a chunk is $2^{32}-1$ (4 GB). As a result of this restriction, the number of elements in a chunk cannot be greater than 4 GB (The HDF Group, no date). And it is necessary to consider the datatype size of each element.

According to (Collette, 2013). The process of picking a chunk shape is a trade-off between the following three constraints:

- Larger chunks for a given dataset size reduce the size of the chunk B-tree, making it faster to find and load chunks.
- Since chunks are all or nothing (reading a portion loads the entire chunk), larger chunks also increase the chance of reading data into memory that won't be needed.
- The HDF5 chunk cache can only hold a finite number of chunks. Chunks bigger than 1 MiB don't even participate in the cache.

7.1.1 Tests performed

for the above, the choice of storage layout and internal organization of the HDF5 file is not a trivial decision and required research and a battery of tests to allow for conclusions. For performing this analysis, we set up a test protocol considering three different dataset sizes, based on the number of columns: L, XL, and XXL all of them subsets of the Source-Dataset presented in Table 7-1.

Table 7-1 - Datasets used for read/write tests with size on disk

Dataset	Rows x Columns.	Size on disk in Bytes	GB
200K (L)	2k x 200k	400 100 352	0,37
600K (XL)	2k x 600k	1 200 004 000	1,12
1000K (XXL)	2k x 1000k	2 000 006 136	1,86

7.1.2 Write Operations over the HDF5 Source-Dataset

These tests were carried out essentially at the expense of successive parallel write operations over the Source-Dataset and all times presented are the average of the runs performed.

The noteworthy fact is that of the total data loading time, about 60% of the time is spent extracting the data from the text file, 40% transforming it into memory and extraordinary fact less than 0.5% is consumed preserving the dataset in the HDF5 file Table 7-2.

As described in the previous section the write is done in a single write operation which has proven to be extremely efficient as it typically takes less than 0.5% of the total load time, regardless of the data block size and chunking layout.

Table 7-2 - How is the time consumed?

	Extract	Transform	Load (Write)	Total
200K (L)	81,2%	18,5%	0,3%	100%
600K (XL)	59,2%	40,7%	0,1%	100%
1000K (XXL)	59,5%	40,3%	0,2%	100%

Table 7-3 - Chunking layout tested

Layout	Meaning
Row chunk	Equals the dimension of the entire row, regardless of its dimension
Best Fit chunk	Is a chunk whose dimension perfectly fits the defined HDF5's read cache size
Contiguous	Don't use chunks
Auto-chunk	Gives HDF5 the responsibility for choosing the dimension

Table 7-3 describes the different options tested. An additional explanation is needed as to “Best Fit chunk” which depends on prior knowledge of the cache size defined at HDF5 installation. This parameter can be

obtained by accessing the low-level API for h5py, whose H5D module provides access to the low-level HDF5 dataset interface using the `get_access_plist()` and `get_cache()` functions (Andrew Collette and contributors, no date). The value obtained was 1048576 bytes. With this information and knowing the dimension in bytes of each row of the different datasets used, it is possible to systematize the different possibilities in Table 7-4.

Table 7-4 - Best Fit chunk estimation per dataset

Dataset	Allocation	Chunk size in bytes	Cache in bytes	Bytes wasted on reading
L	5 x 200k	1 000 005	1 048 576	48 571
XL	1 x 600k	600 001	1 048 576	448 575
XXL	1 x 1000k	1 000 001	1 048 576	48 575

L and XXL have minimal waste, in the case of L per resource to include 5 rows per chunk, but XL is heavily penalized, it wastes about half the cache size.

It was verified in these tests an excessive variability of the times between similar runs. the workload and I/O conditions of the cluster influenced the times obtained mainly in the L tests, even so, and given that in the portion concerning the write operation, these are very low times, (all below 10 seconds) Table 7-5 the focus has to be on small differences in seconds and here are two layouts that stand out negatively; Row chunk for datasets with less than 1000K columns and Auto-chunk with such poor performance that it was abandoned after the first test.

Table 7-5 - Average time to write an entire block of data on HDF5 file in seconds

Layout	200K (L)	600K (XL)	1000K (XXL)
Row chunk	4,91	2,08	1,82
Best Fit chunk	1,27	2,08	1,82
Contiguous	3,87	1,24	5,98
Auto-chunk	7,12		

According to (Collette, 2013) the reason the automatically generated chunks are “square” in N dimensions is that the auto-chunker doesn't have any knowledge of how the information will be accessed, and bet on common usages, such as image storage. It's ideal for those who just want to compress a dataset and don't care about details, but inadequate when specific time-critical access patterns are a concern.

Regarding the use of chunking in write operations, except for Auto-Chunk, there are no major discrepancies depending on the size of the dataset and the chunking used.

Not using chunking at all is not heavily penalized on write operations and may even be faster on 600K column datasets. Best Fit chunk is the most difficult to use and is probably the best solution for some sizes, where it is easier to adjust the dimension to the HDF5 cache size.

7.1.3 Read Operations over the HDF5 Source-Dataset

The component or reading of this test uses the 2nd step of LAID, the Disjoint Matrix generation. The reading pattern of this algorithm over the Source-Dataset is every row is compared with all others, which requires full row reading.

Table 7-6 and chart of Figure 7-1 compares the time consumed reading over different HDF5 Storage layout options, the Auto-chunk option performs so badly even on a small dataset that was abandoned for the larger ones.

Table 7-6 - Read operations versus HDF5 Storage layout (time in seconds)

Dataset	Row Chunk	Best Fit Chunk	Contiguous	Auto-chunk
200K (L)	183,24	174,05	168,37	962,98
600K (XL)	307,20	307,20	264,03	
1000K (XXL)	479,48	479,48	384,34	

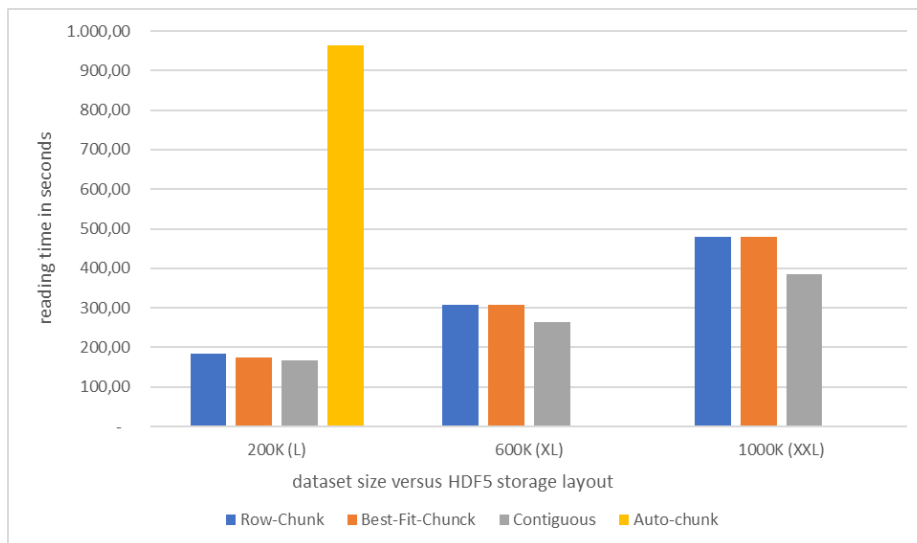


Figure 7-1 - Read operations versus HDF5 Storage layout (time in seconds)

Conclusions can be drawn from due to this pattern:

Chunking is an option to be considered in no way an obligation. Even for the best chunking option, given the read cache size defined in the HDF5 installation and the read pattern used by the algorithm, it is observed that the Storage Layout Contiguous systematically allows greater speed and access.

This speed is hardly distinctive in smaller datasets (L) but gradually becomes more significant as the chart of Figure 7-2 demonstrate.

Contiguous is the best storage layout for larger datasets and indicates a linear relationship between reading time and the number of columns in the dataset.

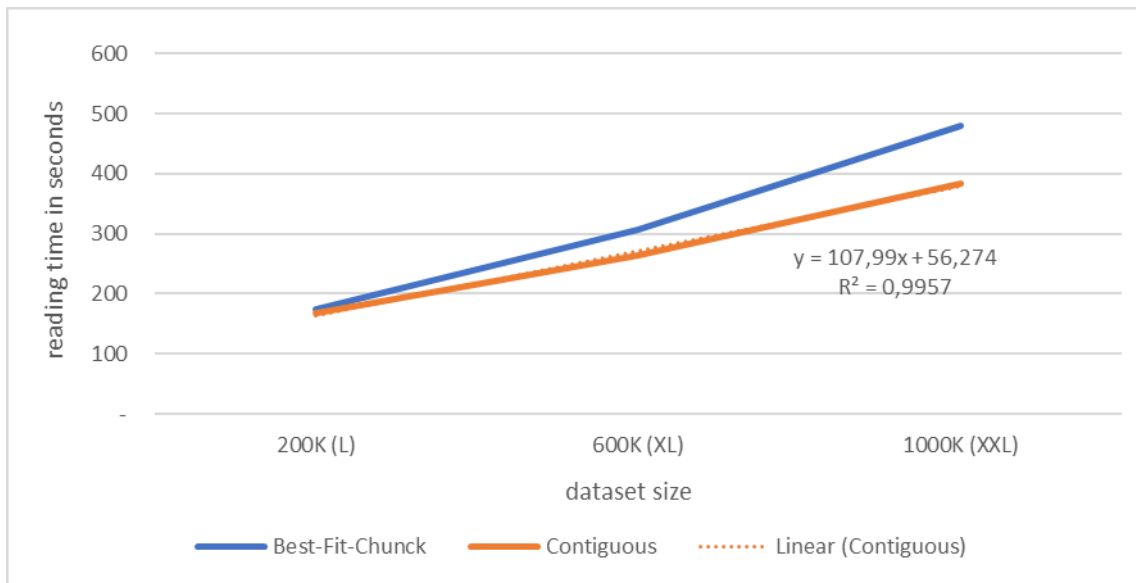


Figure 7-2 - Best fit chunk and contiguous layout comparison for reading workload in seconds

7.1.4 Read and write operations over the DM-Dataset

The tests described above were sufficiently elucidative, so specific tests were not carried out for reading or writing the DM-Dataset. Given the reading pattern of the step 3 algorithm (whole rows and columns) the recourse to any form of chunking for this dataset is a mistake.

7.2 Strategy for HDF5 files and datasets

There doesn't seem to be an advantage in storing class values in the rightmost columns of the dataset, it seems preferable to use another dataset stored in the same HDF5 file. This conclusion does not refer to space-saving, but only to the ease of handling and addressing the respective columns. Notice that the addressing between the two datasets is still maintained by row index.

The DM-Dataset should be kept isolated in a separate HDF5 file, as no advantage was identified in keeping it in the same file as the Source-Dataset. on the contrary, it is more manageable if kept separate. For example, if it is necessary to delete it. The same can be said for any other derived datasets.

When a new fixed-length Dataset is created, HDF5 needs to be informed about the shape i.e., dimensions of both axes. For the Source-Dataset these dimensions are known in advance and are not a problem. But for DM-Dataset only the dimension in columns is known, the same as the Source-Dataset (1000k in the example), the number of rows needs to be estimated.

To estimate it, according to (Cavique *et al.*, 2013) the maximum number of comparisons is given by the expression $n(n-1)/2$ which for $n = 2000$ (observations) means 1 999 000 interactions, therefore, in the worst

case, if all rows are labeled with a different class value, the algorithm will compare all n observations and get that same number of disjoint rows that would need to be preserved on disk.

It is also known that interactions between lines of the same class do not generate disjoint lines and that in the Source-Dataset there are only two different class values (0 and 1), so for this example and because there are 1 700 rows of class 0 and 300 rows of class 1 the maximum number of disjoint rows is given by the expression $1\,700 \times 300 = 510\,000$. This number can even be reduced if there are redundant rows that can be discarded.

In this sample, there are no redundant observations, so it is necessary to consider all 510 000 disjoint rows. Considering the 1000k columns and the one-byte data type, it turns out to store approximately 475 GB on disk which is a large dataset that requires non-negligible storage space whose growth is linear (coefficient of determination = 1) Figure 7-3.

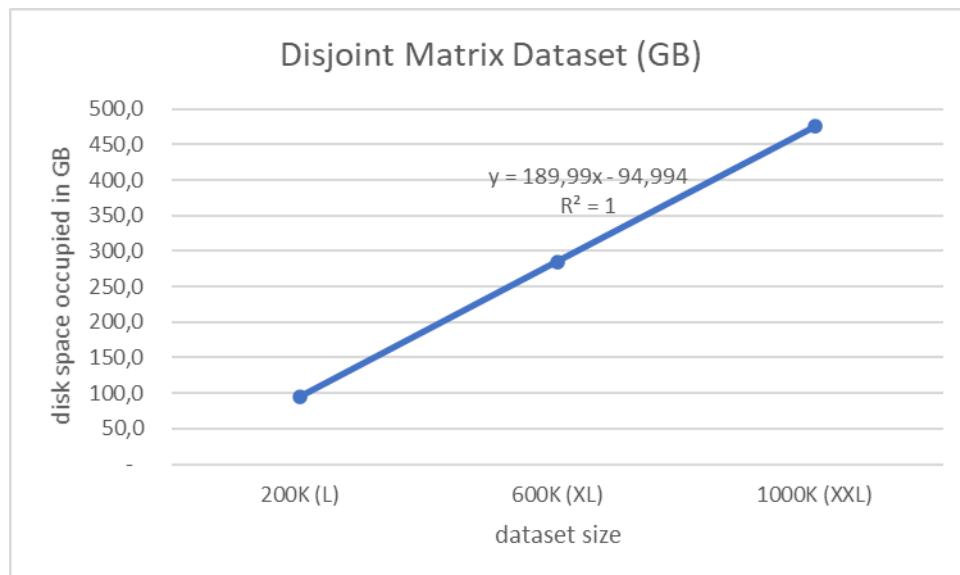


Figure 7-3 - Disk size estimate for DM-Dataset

In the (Apolónia and Cavique, 2019a) experiment, disk space was never a concern because it was done on a single machine with enough disk space. In this case in an HPC environment, with other users competing for the same resources and with a nominal disk quota of 20GB, the space needed to store the DM-Dataset can be a concern, which is why valuable time was spent trying to reduce this size.

7.3 Attempts to reduce the disk size of the DM-Dataset

To evaluate and test how to reduce it, three different solutions described below were tested and discarded. The option for using resizable datasets and abandoning fixed-length was also evaluated, and discarded, as it requires the mandatory use of chunking in the DM-Dataset, which is contraindicated as referred above.

7.3.1 Dataset compression

Following the recommendations (Collette, 2013),(The HDF Group, 2015) the dataset compression was tested, using the *gzip* filter, because GZIP compression is the simplest and most portable compressor for HDF5. however, the use of this filter through h5py and the *mpio* driver requires the use of MPI cooperative mode.

MPI Collective Communication and cooperative mode is a topic by itself, its study and usage implications go far beyond the use of compression. because of this, the option taken was to abandon the *mpio* driver and instead use the native driver which blocks the possibility of parallel use and therefore is not part of the solution, but it was tested anyway.

Compression is supposed to be transparent, data is read and written normally, However, this comes at a cost, compression is a CPU-intensive process where the trade-off is that the most compressed data usually requires the most work to compress/decompress and the longer it takes.

The GZIP compressor offers a range of compression levels from 1 to 9 where 1 offers the fastest compression speed but at a lower ratio, and 9 offers the highest compression ratio but at a slower speed. with a default of 4, which was used as a compromise between the two factors.

The result was entirely disappointing, notice in Figure 7-4 the need to use a logarithmic scale to represent the time in seconds of both compressed and uncompressed runs for only the first 39 600 rows.

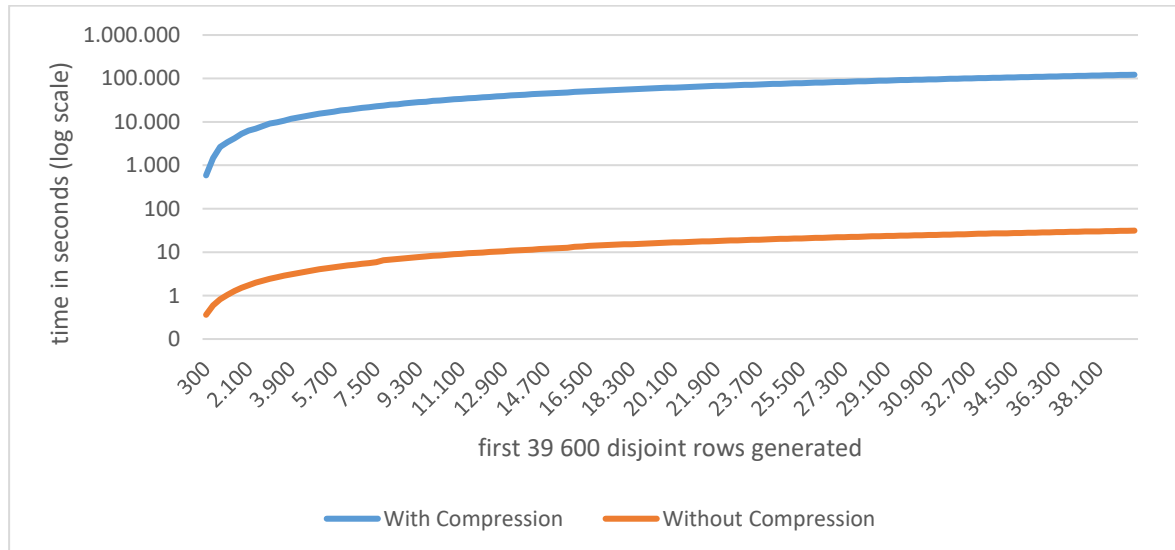


Figure 7-4 - DM-Dataset generation with and without compression

Part of this immense time can be explained by Auto-chunking is also enabled when using compression if a chunk shape is not manually specified. Anyway, this path was abandoned for not allowing parallelization.

7.3.2 Remove empty columns, the "redux" approach

Considering that approximately 31% of the Source-Dataset columns do not contain any value and therefore do not contribute anything to the solution. getting rid of these columns means an equal percentage of storage reduction. of the DM-Dataset which for the XXL source reaches a non-negligible 147 GB.

So, this approach was tested, first, the unvalued columns were marked, (which is a quick task), but removing them from the dataset is a heavy task, the method followed was to copy the unmarked columns to a new dataset. this process took approximately 6 hours for the L dataset, which makes this approach inadequate.

7.3.3 The data type option

From the 2019 experience (Apolónia and Cavique, 2019a), one doubt stands out, why choose a one-byte data type for data columns? when they will only contain only two possible values 0 and 1. Considering a data type that requires only 1 bit per cell would mean a saving of approximately 87% which is a huge difference and justifies revisiting this subject.

The HDF5 library implements an object-oriented model of datatypes. organized as a logical set of datatype classes. Each datatype class defines a format for representing logical values as a sequence of bits. Also defines a set of predefined datatypes, corresponding to commonly used storage formats. See Figure 7-5.

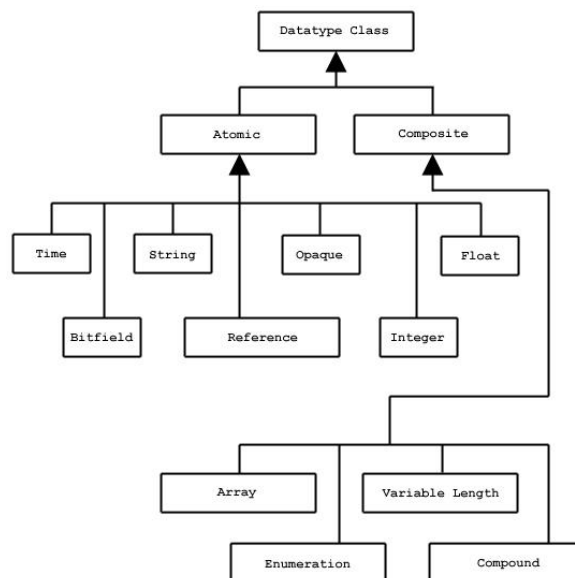


Figure 7-5 - HDF5 Datatype Classes and their properties

HDF5 datatype API provides methods to create datatypes of different datatype classes so a user program can derive types with custom values for the properties and for example “can create a datatype to describe a 6-bit integer” (The HDF Group, no date), so there seems to be no limitation regarding HDF5, but “when transferring

data (e.g., a read or write), the data elements of the source and destination storage must have compatible types. As a general rule, data elements with the same datatype class are compatible, while elements from different datatype classes are not compatible”.

On the other side is Python which defines only one type of a particular data class (only one integer type, one floating-point type, etc.), and the Python approach to Boolean values as two constant objects "True" and "False". They are used to represent truth values in numeric contexts, they behave like the integers “1” and “0”, respectively (Python Software Foundation, no date). meaning Python's default implementation of bool is as a subclass of int.

However, the h5py interface does not rely on python datatypes, but on NumPy, as we saw in section 4.4.2.5 and NumPy, following C rules, takes a different approach, there are 24 new fundamental Python types to describe different types of scalars. These type descriptors are mostly based on the types available in the C language (NumPy Developers, no date). For example, in NumPy, the *bool_* type is not a subclass of the *int_* type moreover is not even a number type and is stored as a byte. So, there is no advantage in choosing it, in terms of memory and storage space savings.

the investigation of the NumPy C-API, despite allowing the creation of its own types, did not allow much progress, that is why, for this purpose, there are no alternatives to *int*, also the smallest platform-independent type available is indeed `numpy.int8` (8 bits). (NumPy Developers, no date). In conclusion, the most suitable data type out of the box is effectively `numpy.int8`: 8-bit signed integer (-128 to 127) or `numpy.uint8`: 8-bit unsigned integer (0 to 255).

7.4 Discussion

It was confirmed that HDF5 is an excellent alternative to store the datasets needed in a LAID analysis. Although it has no performance implications, separating the datasets into different hdf5 files turns out to be a good option, as it allows for better file handling (copy, zipping, deleting from the OS UI, etc.)

It is also a good idea to store the dataset sorted by class values. Through NumPy arrays it is simple to perform this sorting operation even before storing in HDF5; therefore, this part of the problem must be carefully evaluated from the beginning to avoid pitfalls.

The choice of the datatype for the information, as well as the storage layout, must be carefully made and resist the temptation of using HDF5 out of the box. mainly avoid the Auto-chunk option enabled, because the format is automatically chosen by HDF5 it will almost certainly not be adjusted to the reading pattern of the algorithm used and this can lead to considerably degraded performance. From all the above, the preferred layout strategy is not to use chunking at all, opting for contiguous, even so, and for the Source-Dataset chunking is always an option, but it must be correctly evaluated.

8. Computational Experiments and Results

8.1 Method and Action Plan

To develop the code and conduct the computational experiments, a 7-phase plan (called T0 to T6) was defined, providing maximum interactivity where the feedback from the previous phase feeds the next phase, as recommended in the Interactive design cycle of the DSR methodology referred to in Annex 11.1.

Due to the lack of previous experience in HPC, parallel computing, and HDF5, T0 was reserved for the acquisition of the essential knowledge and setting up the environment. T1 and T2 were intended for the experience of loading the original dataset, with parallelization, as well as studying layout alternatives, namely chunking. T3 allowed conduct a series of reading tests, based on parts of the LAID algorithm, using datasets with a different number of columns, and allowed us to draw the first conclusions. T4 and T5 phases were reserved to operationalize all parts of the algorithm and study parallelization alternatives. T6 for last adjustments and finally for solution tests.

The work on INCD infrastructure was performed between 2021-03-28 and 2021-09-07, during this time 317 runs jobs are performed of which 267 (84%) were completed successfully. these jobs occupied a total of 312 CPU hours, involving a maximum of 10 CPUs per job and an average of 2 CPUs per job, running on a maximum of 2 computational nodes.

```
1  #!/bin/bash
2
3  #SBATCH --job-name=LaidPoS          # submit_check.sh
4  #SBATCH --time=24:24:24           # max time
5  #SBATCH --ntasks=5                # Number of MPI ranks (total number of tasks)
6  #SBATCH --nodes=1                 # Run all processes on a x nodes
7  #SBATCH --partition=hpc           # CIRRUS-A (Lisbon)
8
9  echo "JobId"                      = $SLURM_JOBID"
10 echo "Date/Time Start"            = $(date +%F %T,%N)"
11 echo "Hostname"                   = $(hostname -s)"
12 echo "Working Directory"          = $(pwd)"
13 echo ""
14 echo "Number of Nodes Allocated"   = $SLURM_JOB_NUM_NODES"
15 echo "Number of Tasks Allocated"  = $SLURM_NTASKS"
16 echo "Number of Cores/Task Allocated" = $SLURM_CPUS_PER_TASK"
17 echo "SLURM_ARRAY_TASK_ID"        = $SLURM_ARRAY_TASK_ID"
18 echo "Running LAID parallel with serial code on $SLURM_CPUS_ON_NODE CPU cores"
19
20 module purge                       # clean and load environment
21 module load hdf5/1.12.0
22 module list
23 echo "python version"
24 python --version
25
26 rm laidp_aux.h5
27
28 mpirun python laid_serial_6b.py     # job payload
29 echo ""
30 echo "JobId finished"              = $SLURM_JOBID"
31 echo "Date End"                    = $(date +%F %T,%N)"
```

Figure 8-1 - Bash script for launching jobs through mpirun

Figure 8-1 presents a bash script used for launching python code jobs. It defines the parameters necessary for slurm to know what resources are being requested and allows its scheduling, clean, and load the environment with the necessary modules. eliminate auxiliary datasets, echo parameters to document the run, and finally launch the job using the mpirun command. An example of the output is shown in annex 11.3.

All Python code has been provided with performance counters for benchmarking whose information was cross-referenced with the start and end date-time of each job. therefore, there is high confidence in the observed time, however, Some caution is needed in analyzing the reported times. On several occasions, it was found that the same code and data running in different periods returns the same result but with discrepancies in the time spent which indicates that the job environment is not fully isolated and is affected by cluster operational conditions, it could be concurrent workloads, network latency or other cause, It would be important to identify the reasons, but the conditions for this analysis are not met, nor is it part of the scope of this work.

8.2 Serial, Hybrid and Parallel Runs

Figure 8-2 illustrates and synthesizes the three forms of runs used in this work. The simplest model follows the sequential execution, is represented on the left side of the figure, the center represents a hybrid form, where part of the processing is serial, and part parallelized, and the parallel model is represented on the right side.

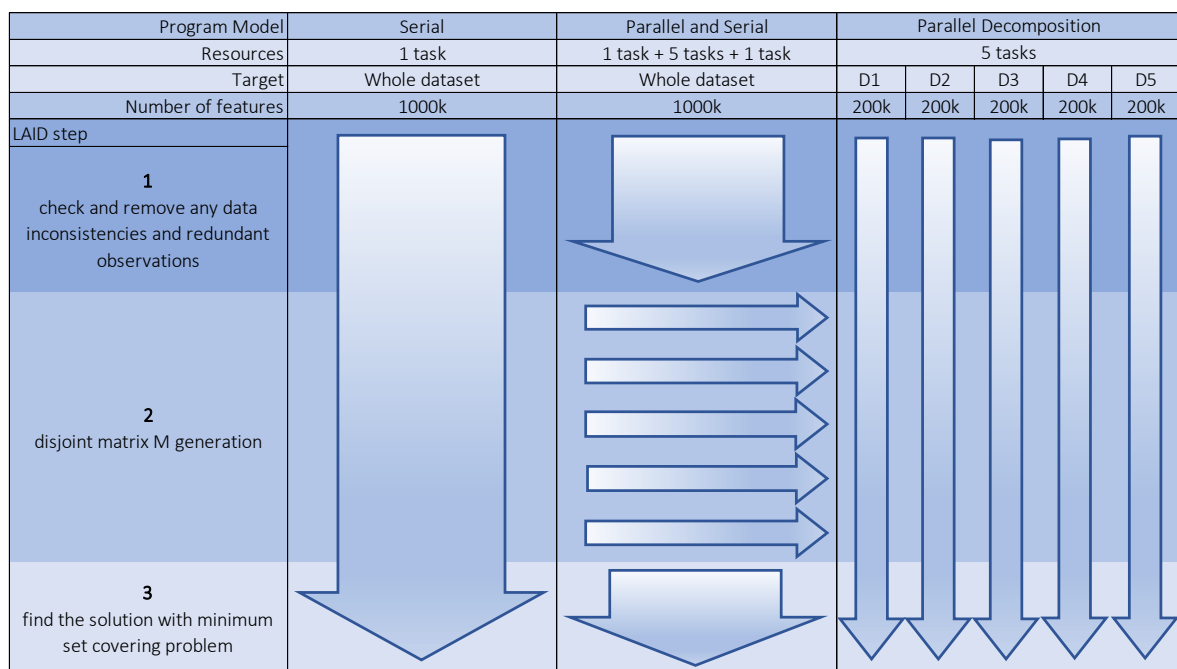


Figure 8-2 - Serial, Hybrid and Parallel Runs

8.3 Serial Program Model

This solution is largely a repetition of the 2019 procedure (Apolónia and Cavique, 2019a), it was important to carry out because its times are an interesting comparison with that past experience and mainly serves as a reference for current parallelization experiments. The average time consumed by each step is described in Table 8-1.

Table 8-1 – Average time spent running Serial LAID steps by step

LAID Step		T0	
1	Check and remove inconsistency and redundant observations	1.254,96	
2	Disjoint matrix generation	first-pass	N/A
		second-pass	1.352,21
3	Find a solution with minimum set covering problem	6.221,88	
total in seconds:		8.829,06	
total in minutes:		147,15	

This is approximately half the time reported by (Apolónia and Cavique, 2019a), however, there are too many differences between the two experiments, different datasets, and environments, the HPC effect, etc. to draw secure conclusions.

8.4 Hybrid of Serial and Parallel Programming Models

In this solution, only step 2, DM-Dataset generation, is parallelized through a horizontal data partition. As we saw in section 5.3, the algorithms from steps 1 and 3 were not converted due to the non-use of the MPI cooperative mode.

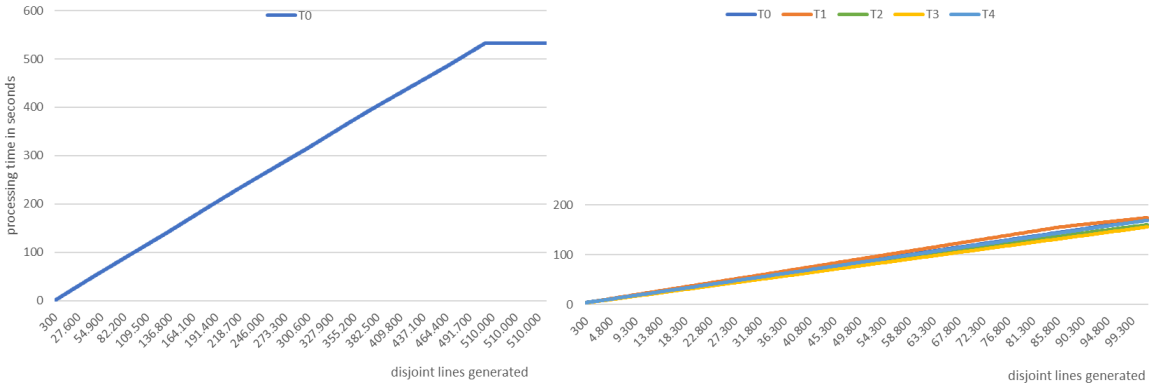


Figure 8-3 - DM-Dataset generation, Comparison of the serial version against 5 parallel tasks

For this specific dataset and due to its characteristics, described in section 6, the maximum number of parallel tasks is 1700, each performs 300 comparisons, or even better if we swapped the order of the dataset classes, 300 parallel tasks, each performing 1700 comparisons. More realistically in this experiment, only 5 parallel tasks were used so, the 1700 lines were divided by 5 and each job perform 340 x 300 comparisons with the results described on the chart in Figure 8-3.

Table 8-2 - Time spent running Hybrid approach

LAID Step		Serial	T0	T1	T2	T3	T4
1	Check and remove inconsistency and redundant observations	1.254,96					
2	Disjoint matrix generation	first-pass	407,83				
		second-pass	444,43	444,43	441,50	431,18	431,16
3	Find a solution with minimum set covering problem	6.221,88					
total in seconds:		8.329,11					
total in minutes:		138,82					

As expected, for the parallel version the difference is appreciable, a speedup of 3,04 with an efficiency of 0,6, this value can even be further improved by fine-tuning the buffer size.

Table 8-3 - Performance Metrics for Hybrid approach

T _{serial}	T _{parallel}	Processors	Speedup Sp	Efficiency E
1.352,21	444,43	5	3,04	0,61

8.5 Parallel decomposition through vertical data partition

The previous solution, although faster than the serial solution, is not practical. A much better solution is to run several jobs in parallel of all the serial code, that is, parallelize the execution of the serial version. This way, each job will process subsets of the entire dataset, for example, 200k column blocks. Then after the parallel operation, a final run could be performed to reduce the result of the various solutions found. This way the bottleneck constituted by step 3, which was not fully resolved, has less impact.

Table 8-4 - Time spent running Parallel Decomposition

LAID Step		T0	T1	T2	T3	T4	
1	Check and remove inconsistency and redundant observations	32,28	32,60	32,81	33,11	32,98	
2	Disjoint matrix generation	first-pass	211,71	212,91	206,97	211,94	212,05
		second-pass	237,44	240,05	236,69	242,38	237,51
3	Find a solution with minimum set covering problem	1.044,77	988,40	956,70	1.068,36	918,17	
total in seconds:		1.526,19	1.473,95	1.433,17	1.555,79	1.400,71	
total in minutes:		25,44	24,57	23,89	25,93	23,35	

Comparing and evaluating the running times of LAID algorithms for 1000k features using Serial against parallel version, where the same number of features is subdivided into 5 batches of 200k takes 147.15 minutes for serial to solve against 25.93 minutes on parallel version. Table 8-5 resumes this evaluation.

Table 8-5 - Performance Metrics for Parallel over Serial

T _{serial}	T _{parallel}	Processors	Speedup Sp	Efficiency E
147,15	25,93	5	5,67	1,13

8.6 Discussion

Table 8.6 presents a comparative board of the three approaches described above, for the last run parallel T1 was chosen because it has the worst (longest) performance of the third run.

Some caution is needed when comparing this outcome. Parallel T3, (selected because it was the task with the worst performance), could not be directly compared with others as only handled 200k only the total time could be compared with no restrictions. And as expected the fully parallel version is much faster than any of the others.

Table 8-6 - Comparative board of three approaches Serial/Hybrid/Parallel decomposition

	LAID Step	Serial	Hybrid	Parallel T3
1	Check and remove inconsistency and redundant observations	1.254,96	1.254,96	33,11
2	Disjoint matrix generation	first-pass	N/A	407,83
		second-pass	1.352,21	444,43
3	Find a solution with minimum set covering problem	6.221,88	6.221,88	1.068,36
total in seconds:		8.829,06	8.329,11	1.555,79
total in minutes:		147,15	138,82	25,93

The result of the solution considering all columns of the source dataset is only the column with index 100004. The reason is that this column is exactly equal to the class column. Any quality evaluation would reveal 100% of Accuracy. The conclusion is that it makes no sense to evaluate this specific case.

There are also two notes regarding the running of the serial version in parallel.

First changes to the code of the serial supposedly should be simple, since the partition of the columns of the Source-Dataset is made, it is necessary to ensure that the rows generated from the DM-Dataset are equally separated to be processed by different tasks. to achieve this, the creation of 5 datasets within the same HDF5 file was attempted. but it was not possible due to errors in the Lustre file system on their creation. The same

happened when trying the operation on separate HDF5 files. In no case was the creation operation possible when executed in code running in parallel.

Due to the number of components and layers involved troubleshooting attempted using specialized sites such as stackoverflow.com did not allow us to isolate and identify the cause and resolve the issue, in the list of suspects are the main components: h5py/hdf5, the MPI implementation, and even Lustre.

To overcome the problem, It was necessary to create the files in advance using serial mode and leave the job of filling them out by parallel tasks. Although this solution is not entirely satisfactory, the results are quite good, but more study and testing will be needed for this specific issue.

Second, after testing with 5 parallel tasks and getting very acceptable times, the next step was to ask why not subdivide into even smaller batches and presumably get even shorter runtimes? What was tried, but this Source-Dataset does not allow subdivision below 200k, because as mentioned in section 6.3 each block of 200k is not uniform, and when treated separately, the second part of each block presents a high number of redundant observations 1998 for a total of 2000 which would introduce error in the final solution.

9. Conclusions

9.1 Contributions

The research work described in this master's dissertation is part of a sequence of previous works produced by different authors, to contribute to the usage of the Logical Analysis of Inconsistent Data methodology in the problem of selecting features in highly dimensioned datasets. A contribution that constitutes yet another “brick” in the much broader construction of knowledge for the effective solution of a current and increasingly relevant problem.

The approach followed is in line with current trends towards computationally intensive algorithms increasingly considered a paradigm shift in scientific environments, where working data is stored on HDF5 format on disk, and running the algorithms in parallel in a cloud High-Performance Computing environment.

The objective of discovering how parallel processing can reduce processing times for the end-user and describing how the HPC paradigm applies to this problem led to the decomposition of the problem into several parts that we tried to answer.

To do this the Design Science Research methodology was followed, a Systematic Literature Review was carried out to compile a theoretical background that supports this research.

The key constituent parts of the HPC environment used have been described and documented, hoping that they will be useful in future work that will certainly take place.

Also, about the sample dataset used, useful information and data insights were extracted that can be useful in future situations to which is added a work of analysis and comparison of the best strategies and layouts for storing information in HDF5, meeting the reading patterns required by the different steps of LAID.

It has been fully confirmed and demonstrated that HDF5 is an excellent way to store the dataset used in a LAID analysis. It presents excellent performance for reading and writing. But even so, and especially in parallel use, it was also demonstrated that parallel code must implement ways to avoid overload to avoid degrading performance. The continuous layout was also indicated as the most suitable for the datasets used and the conditions where the alternatives can be used were described.

As for the parallelization of the LAID algorithms, code developed in previous works was adapted and newly developed as needed. Which was later tested on INCD HPC environment, following a testing protocol to produce the results presented. To highlight the reduction from 140 minutes of the serial version to 24 minutes of the parallel version of a dataset with 1000k features, which represents an acceptable time and an 82% reduction of the previous time.

Yet regarding this topic of parallelization, not all goals were achieved. It was left to do the parallelization of the LAID step 3, but even in this case, it was possible to describe what is believed to be the solution which will certainly go through the use of the MPI collaborative mode.

To the constituents of the proposed artifact, which as stated are the parallel code, the organization of HDF5 files and datasets, the orchestration of the process. It also added an interesting set of lessons learned and guides that enrich it and justify the conviction about its usefulness.

9.2 Limitations

The absence of essential knowledge about essential concepts such as parallel programming, HPC, and HDF5 was an initial limitation, obviously surmountable, but it required substantial effort and consumed precious time.

The use of HDF5 via h5py, in the subsequent phases of the LAID analysis, especially in the creation of the DM-Dataset, reveals some weaknesses, mainly related to some lack of flexibility that comes down to the need to know in advance the number of rows to create a dataset with fixed dimensions. The alternative of using resizable datasets also reveals little flexibility by forcing the use of chunking. In contrast, the I/O speed is very good, the bottlenecks found to result from excessive write operations overhead in multitasking mode, avoidable with the introduction of a buffer that limits the number of write operations, in favor of less writing but with more data at a time. This seems to be one of the golden rules.

Attempts to reduce the DM-Dataset disk size using compression have been disappointing.

The impact of overall workload conditions on the cluster is an issue that affects runtimes and complicates the act of concluding.

Despite the potential of the Python, NumPy, h5py, HDF5 stack, especially the ease of use, details such as the impossibility of defining a custom data type justify considering the choice of other elements for this stack that allow a greater level of control even with fewer out-of-the-box features.

9.3 Future Work

It would be very interesting to continue the parallelization effort of the LAID steps. Using the MPI Collective Communication Mode and how this use impacts favorably or unfavorably on the execution of LAID algorithms, with an obvious emphasis on LAID's step 3.

It would also be interesting to search for a fast and efficient way to remove irrelevant columns while keeping all the original column index information intact.

It might also be interesting to study the use of the HDF5 Virtual Dataset (VDS) feature to allow the "merging" of datasets created by different parallel tasks into separate files.

From a systems management perspective and about the future use of the LAID methodology in an enterprise context, it would be interesting to study ways of operationalizing it in a CLOUD as a Service context. Considering the essential points such as financial, technical, and operational requirements without forgetting the need for security and confidentiality that this kind of information requires.

10. References

- Aaron, C. X. *et al.* (2020) 'Mapping Datasets to Object Storage System', *EPJ Web of Conferences*, 245, p. 4037. doi: 10.1051/epjconf/202024504037.
- Agrawal, A. and Choudhary, A. (2016) 'Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science', *APL Materials*, 4(5). doi: 10.1063/1.4946894.
- Agrawal, R. and Srikant, R. (1994) 'Fast Algorithms For Mining Association Rules In Datamining', *International Journal of Scientific & Technology Research*. Edited by J. B. Bocca, M. Jarke, and C. Zaniolo, 2(12), pp. 13–24. Available at: citeseer.ist.psu.edu/agrawal94fast.html.
- Andrew Collette and contributors (no date) *h5py - HDF5 for Python*. Andrew Collette and contributors. Available at: <https://docs.h5py.org/en/stable/index.html> (Accessed: 12 July 2021).
- Apolónia, J. and Cavique, L. (2019a) 'Seleção de Atributos de Dados Inconsistentes em ambiente HDF5 + Python na cloud INCD', *Revista de Ciências da Computação*, (14), pp. 85–112. doi: 10.34627/rcc.v14i0.
- Apolónia, J. and Cavique, L. (2019b) *Seleção de atributos utilizando a Análise Lógica de Dados Inconsistentes (LAID)*, *Repositório institucional da Universidade Aberta (UAb)*. Available at: <http://hdl.handle.net/10400.2/8122> (Accessed: 6 May 2021).
- Asaadi, H. R., Khaldi, D. and Chapman, B. (2016) 'A comparative survey of the HPC and big data paradigms: Analysis and experiments', *Proceedings - IEEE International Conference on Cluster Computing, ICC3*, pp. 423–432. doi: 10.1109/CLUSTER.2016.21.
- Azevedo, A. and Santos, M. F. (2008) 'KDD, semma and CRISP-DM: A parallel overview', *IADIS European Conference Data Mining*, pp. 182–185. Available at: <http://recipp.ipp.pt/handle/10400.22/136%0Ahttp://recipp.ipp.pt/bitstream/10400.22/136/3/KDD-CRISP-SEMMA.pdf>.
- Barrand, G. *et al.* (2019) 'HDF5 and row-wise ntuple in analysis tools in Geant4 10.4.', *EPJ Web of Conferences*, 214, pp. 1–7. Available at: <http://10.0.4.27/epjconf/201921402009>.
- Blelloch, G. E., Simhadri, H. V. and Tangwongsan, K. (2012) 'Parallel and I/O efficient set covering algorithms', *Annual ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 82–90. doi: 10.1145/2312005.2312024.
- Bolis, A. *et al.* (2016) 'An adaptable parallel algorithm for the direct numerical simulation of incompressible turbulent flows using a Fourier spectral/hp element method and MPI virtual topologies', *Computer Physics Communications*, 206, pp. 17–25. doi: 10.1016/j.cpc.2016.04.011.
- Boros, E. *et al.* (1997) 'Logical Analysis of Numerical Data', *Mathematical Programming. Springer*, 79(October), pp. 163–190.
- Brankovic, A., Hosseini, M. and Piroddi, L. (2019) 'A Distributed Feature Selection Algorithm Based on Distance Correlation with an Application to Microarrays', *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(6). doi: 10.1109/TCBB.2018.2833482.
- Brett, A. (2016) *Tokamak data mirror for JET and MAST data*, *EUDAT-CDI*. Available at: <https://www.eudat.eu/use-cases/tokamak-data-mirror-for-jet-and-mast-data> (Accessed: 1 October 2021).
- Cavique, L. *et al.* (2013) 'A feature selection approach in the study of azorean proverbs', in *Exploring Innovative and Successful Applications of Soft Computing*. Exploring. Hershey: IGI Global, pp. 38–58. doi: 10.4018/978-1-4666-4785-5.ch003.
- Cavique, L. *et al.* (2018) 'A biobjective feature selection algorithm for large omics datasets', in *Expert Systems*. doi: 10.1111/exsy.12301.
- Cavique, L., Mendes, A. B. and Funk, M. (2011) 'Logical Analysis of Inconsistent Data (LAID) for a Paremiologic Study', in *15th Portuguese Conference on Artificial Intelligence, EPIA 2011*.
- CentOS Project (no date) *CentOS Linux distribution*. The CentOS Project. Available at:

<https://www.centos.org/about/> (Accessed: 14 August 2021).

Chakravarty, S. and Shekhawat, A. (1992) 'Parallel and serial heuristics for the minimum set cover problem', *The Journal of Supercomputing*, 5(4), pp. 331–345. doi: 10.1007/BF00127952.

Chvatal, V. (1979) 'A Greedy Heuristic for the Set-Covering Problem', *Mathematics of Operations Research*. doi: 10.1287/moor.4.3.233.

Collette, A. (2013) *Python and HDF5*. 2013th edn. Sebastopol, Calif: O'Reilly Media. Available at: <http://shop.oreilly.com/product/0636920030249.do>.

Coutinho, E. F., Paillard, G. and De Souza, J. N. (2014) 'Performance analysis on scientific computing and cloud computing environments', *ACM International Conference Proceeding Series*, (April). doi: 10.1145/2590651.2590656.

Cummings, M. P. and Huskamp, J. C. (2005) 'Grid Computing', *EDUCAUSE review*, 40(6 (November/December 2005)), pp. 116–117. Available at: <https://er.educause.edu/articles/2005/1/grid-computing>.

Dalcín, L., Paz, R. and Storti, M. (2005) 'MPI for Python', *Journal of Parallel and Distributed Computing*. doi: 10.1016/j.jpdc.2005.03.010.

Dong, B., Byna, S. and Wu, K. (2013) 'Expediting scientific data analysis with reorganization of data', in *2013 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, pp. 1–8. doi: 10.1109/CLUSTER.2013.6702675.

Dong, B., Byna, S. and Wu, K. (2014) 'Parallel query evaluation as a Scientific Data Service', *2014 IEEE International Conference on Cluster Computing (CLUSTER), Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, pp. 194–202. doi: 10.1109/CLUSTER.2014.6968765.

Dongarra, J. (2004) 'The boole lecture trends in high performance computing', *Computer Journal*, 47(4), pp. 399–403. doi: 10.1093/comjnl/47.4.399.

Ec.europa.eu (2021) *High Performance Computing, European Commission*. Available at: <https://ec.europa.eu/digital-single-market/en/high-performance-computing> (Accessed: 27 March 2021).

Edosio, U. Z. (2014) 'Big Data Paradigm- Analysis , Application, and Challenges', *Electronics and Telecommunications Research Seminar Series 13th Workshop*, (April).

El-Seoud, S. A. *et al.* (2017) 'Big data and cloud computing: Trends and challenges', *International Journal of Interactive Mobile Technologies*, 11(2), pp. 34–52. doi: 10.3991/ijim.v11i2.6561.

Ertl, C., Frisch, J. and Mundani, R.-P. (2017) 'Design and optimisation of an efficient HDF5 I/O kernel for massive parallel fluid flow simulations'. doi: 10.1002/cpe.4165.

FCT – Fundação para a Ciência e a Tecnologia (2020) 'Portuguese Roadmap of Research Infrastructures 2020 Update', pp. 5–7.

Folk, M. *et al.* (2011) 'An overview of the HDF5 technology suite and its applications', *ACM International Conference Proceeding Series*, pp. 36–47. doi: 10.1145/1966895.1966900.

Gaillard, M. (2017) *CERN Data Centre passes the 200-petabyte milestone, Computing*. Available at: <https://home.cern/news/news/computing/cern-data-centre-passes-200-petabyte-milestone> (Accessed: 1 October 2021).

GNU project and GCC developers (no date) *GNU Compiler Collection*. Available at: <https://gcc.gnu.org/> (Accessed: 2 August 2021).

Han, J., Kamber, M. and Pei, J. (2012) *Data Mining: Concepts and Techniques*. Third Edit. Elsevier Inc.

Hevner, A. R. *et al.* (2004) 'Design Science in Information Systems Research', *MIS Quarterly*, 28(1), pp. 75–105.

- Hevner, A. R. (2007) 'A Three Cycle View of Design Science Research', *Scandinavian Journal of Information Systems*, 19(2), pp. 87–92. doi: <http://aisel.aisnet.org/sjis/vol19/iss2/4>.
- Hey, T., Tansley, S. and Tolle, K. (2009) *The Fourth Paradigm*. 2009th edn. Edited by T. Hey, S. Tansley, and K. Tolle. Microsoft Research. Available at: <http://fourthparadigm.org>.
- Howison, M. et al. (2010) 'Tuning HDF5 for Lustre File Systems', in *Workshop on Interfaces and Abstractions for Scientific Data Storage (IASDS10)*. Available at: <http://search.ebscohost.com/login.aspx?direct=true&db=edsstc&AN=edsstc.1050648&lang=pt-br&site=eds-live&scope=site>.
- Hummel, J. A. (2016) 'gadfly: A pandas-based Framework for Analyzing GADGET Simulation Data', *Publications of the Astronomical Society of the Pacific*, 128(969), p. 114503. doi: 10.1088/1538-3873/128/969/114503.
- Incardona, P. et al. (2019) 'OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers.', *Computer Physics Communications*, 241, pp. 155–177. doi: 10.1016/j.cpc.2019.03.007.
- INCD - Infraestrutura Nacional de Computação Distribuída (no date) *INCD user documentation*. Available at: <https://wiki.incd.pt/books> (Accessed: 3 April 2021).
- Iosup, A. et al. (2011) 'Performance analysis of cloud computing services for many-tasks scientific computing', *IEEE Transactions on Parallel and Distributed Systems*. doi: 10.1109/TPDS.2011.66.
- Kitchenham, B. (2004) 'Procedures for Performing Systematic Reviews', *Keele University Technical Report TR/SE-0401*.
- Kohavi, R. and John, G. H. (1998) 'The Wrapper Approach', Liu H., Motoda H. (eds) *Feature Extraction, Construction and Selection. The Springer International Series in Engineering and Computer Science*, 453, pp. 33–50. doi: 10.1007/978-1-4615-5725-8_3.
- Kohavi, R. and John, H. (1997) 'Wrappers for feature subset selection', *Artificial Intelligence*, 97(1–2), pp. 273–324. Available at: <http://linkinghub.elsevier.com/retrieve/pii/S000437029700043X>.
- Kubica, J., Singh, S. and Sorokina, D. (2011) 'Parallel large-scale feature selection', in R. Bekkerman, M. Bilenko, & J. L. (ed.) *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, pp. 352–370. doi: 10.1017/CBO9781139042918.018.
- Kumar, V. and Minz, S. (2014) 'Feature Selection: A literature Review', *The Smart Computing Review*, 4(3). doi: 10.6029/smartcr.2014.03.007.
- Kurth, T. et al. (2015) 'High-Performance I/O: HDF5 for Lattice QCD', *Proceedings of Science*, Part F1305. doi: 10.22323/1.214.0045.
- Lawrence Livermore National Laboratory (no date) *HPC Training Resources*. Available at: <https://hpc.llnl.gov/training/tutorials> (Accessed: 20 May 2021).
- López, F. G. et al. (2015) *High-dimensional feature selection via feature grouping: A Variable Neighborhood Search approach, Parallel Metaheuristics: A New Class of Algorithms*. doi: 10.1002/0471739383.ch10.
- Masui, K. et al. (2015) 'A compression scheme for radio data in high performance computing', *Astronomy and Computing*, 12, pp. 181–190. doi: 10.1016/j.ascom.2015.07.002.
- McKinney, W. (2010) 'Data Structures for Statistical Computing in Python', *Proceedings of the 9th Python in Science Conference*, 1(Scipy), pp. 56–61. doi: 10.25080/majora-92bf1922-00a.
- Mehta, K. et al. (2012) 'A Plugin for HDF5 Using PLFS for Improved I/O Performance and Semantic Analysis', *2012 SC Companion: High Performance Computing, Networking Storage and Analysis, High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion., High Performance Computing, Networking Storage and Analysis, SC Companion: IEEE*, pp. 746–752. doi: 10.1109/SC.Companion.2012.102.

- Moxey, D. *et al.* (2020) 'Nektar++: Enhancing the capability and application of high-fidelity spectral/hp element methods.', *Computer Physics Communications*, 249, p. N.PAG-N.PAG. doi: 10.1016/j.cpc.2019.107110.
- MPI-Forum (no date) *MPI Standard*. Available at: <https://www.mpi-forum.org> (Accessed: 18 June 2021).
- National Energy Research Scientific Computing Center (no date) *NERSC Documentation*. Available at: <https://docs.nersc.gov/> (Accessed: 15 May 2021).
- NumPy Developers (no date) *Numpy - Numerical Python*. Available at: <https://numpy.org> (Accessed: 30 May 2021).
- Ohio State University (no date) *mvapich2*. Ohio State University. Available at: <https://mvapich.cse.ohio-state.edu> (Accessed: 24 August 2021).
- Open-MPI-Project (no date) *Open MPI*. Available at: <https://www.open-mpi.org> (Accessed: 13 June 2021).
- Open Scalable File Systems, I. and E.-E. O. F. S. (no date) *Lustre® filesystem*. Available at: <https://www.lustre.org/> (Accessed: 5 September 2021).
- Otoo, E., Nimako, G. and Ohene-Kwofie, D. (2012) 'Using chunked extendible array for physical storage of scientific datasets', *Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012*, pp. 1315–1321. doi: 10.1109/SC.Companion.2012.164.
- Pacheco, P. (2011) *An Introduction to Parallel Programming, An Introduction to Parallel Programming*. Elsevier Inc. doi: 10.1016/C2009-0-18471-4.
- Pawlak, Z. (1991) *Rough sets: Theoretical aspects of reasoning about data*, Kluwer Academic Publishers. Boston: Kluwer Academic Publishers. doi: 10.1016/s0967-0661(96)90021-0.
- Peffer, K. *et al.* (2007) 'A design science research methodology for information systems research', *Journal of Management Information Systems*, 24(3), pp. 45–77. doi: 10.2753/MIS0742-1222240302.
- Pentland, A. (2013) 'The data-driven society', *Scientific American*, 309(4), pp. 78–83. doi: 10.1038/scientificamerican1013-78.
- Phan, A. H. and Cichocki, A. (2010) 'Tensor decompositions for feature extraction and classification of high dimensional datasets', *Nonlinear Theory and Its Applications, IEICE*, 1(1), pp. 37–68. doi: 10.1587/nolta.1.37.
- Pourmal, E., Cheng, A. and Aydt, R. (2009) 'HDF5 and netCDF-4: Two Solutions for Data Management Problems Based on One File Format', pp. 4–6.
- Price, D. C., Barsdell, B. R. and Greenhill, L. J. (2015) 'HDFITS: Porting the FITS data model to HDF5', *Astronomy and Computing*, 12, pp. 212–220. doi: 10.1016/j.ascom.2015.05.001.
- Python Software Foundation (no date) *Python*. Python Software Foundation. Available at: <https://www.python.org> (Accessed: 14 April 2021).
- Rastogi, S. and Zaheer, H. (2016) 'Significance of parallel computation over serial computation', *International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016*, pp. 2307–2310. doi: 10.1109/ICEEOT.2016.7755106.
- Reuter, K. and Köfinger, J. (2019) 'CADISHI: Fast parallel calculation of particle-pair distance histograms on CPUs and GPUs', *Computer Physics Communications*, 236, pp. 274–284. doi: 10.1016/j.cpc.2018.10.018.
- Rissino, S. and Lambert-Torres, G. (2009) 'Rough Set Theory - Fundamental Concepts, Principals, Data Extraction, and Applications', *Data Mining and Knowledge Discovery in Real Life Applications*, (February). doi: 10.5772/6440.
- Ritter, M. *et al.* (2011) 'Using HDF5 for Cross Disciplinary Data Archival and Highly Performing Processing of Multi-Variant Observational and Computational Data with Simple or Complex Topology', *Geophysical Research Abstracts*.

- Roffo, G. and Melzi, S. (2017) 'Features Selection via Eigenvector Centrality', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10312 LNCS, pp. 19–35.
- Saeys, Y., Inza, I. and Larrañaga, P. (2007) 'A review of feature selection techniques in bioinformatics', *Bioinformatics*, 23(19), pp. 2507–2517. doi: 10.1093/bioinformatics/btm344.
- Saif, S. and Wazir, S. (2018) 'Performance Analysis of Big Data and Cloud Computing Techniques: A Survey', *Procedia Computer Science*, 132, pp. 118–127. doi: 10.1016/j.procs.2018.05.172.
- Savelieff, M. (2019) *Next-Generation Sequencing (NGS): Stimulating the Next Generation of Cancer Diagnostics and Treatment*, *technologynetworks.com*. Available at: <https://www.technologynetworks.com/diagnostics/articles/next-generation-sequencing-ngs-stimulating-the-next-generation-of-cancer-diagnostics-and-treatment-319649> (Accessed: 1 October 2021).
- SchedMD (no date) *Slurm Workload manager*. Available at: <https://slurm.schedmd.com/documentation.html> (Accessed: 26 August 2021).
- Sharma, N. and Saroha, K. (2015) 'Study of dimension reduction methodologies in data mining', in *International Conference on Computing, Communication and Automation, ICCCA 2015*. doi: 10.1109/CCAA.2015.7148359.
- Sreepathi, S. et al. (2013) 'SCORPIO: A scalable two-phase parallel I/O library with application to a large scale subsurface simulator', *20th Annual International Conference on High Performance Computing, HiPC 2013*, pp. 443–451. doi: 10.1109/HiPC.2013.6799128.
- Subasi, M. M. and Avila-Herrera, J. F. (2016) 'Logical analysis of multiclass data', *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2016*.
- Talia, D. (2002) 'Parallelism in knowledge discovery techniques', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2367, pp. 127–136. doi: 10.1007/3-540-48051-x_14.
- Talia, D. (2019) 'A view of programming scalable data analysis: from clouds to exascale', *Journal of Cloud Computing*, 8(1). doi: 10.1186/s13677-019-0127-x.
- The HDF Group (2015) *Improving I / O Performance When Working with HDF5 Compressed Datasets*. Available at: <https://support.hdfgroup.org/HDF5/doc/TechNotes/TechNote-HDF5-ImprovingIOPerformanceCompressedDatasets.pdf>.
- The HDF Group (no date) *HDF5 high performance data software library and file format*. The HDF Group. Available at: <https://www.hdfgroup.org/solutions/hdf5> (Accessed: 13 May 2021).
- The pandas development team (no date) *Pandas*. Available at: <https://pandas.pydata.org/> (Accessed: 3 June 2021).
- VanderPlas, J. (2017) *Python Data Science Handbook*. O'Reilly Media, Inc. Available at: <https://jakevdp.github.io/PythonDataScienceHandbook/>.
- Vecchiola, C., Pandey, S. and Buyya, R. (2009) 'High-performance cloud computing: A view of scientific applications', *I-SPAN 2009 - The 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pp. 4–16. doi: 10.1109/I-SPAN.2009.150.
- Verleysen, M. and François, D. (2005) 'The curse of dimensionality in data mining and time series prediction', *Lecture Notes in Computer Science*, 3512(June), pp. 758–770. doi: 10.1007/11494669_93.
- Verros, S. A. et al. (2017) 'Computing spatial correlation of ground motion intensities for ShakeMap', *Computers and Geosciences*, 99(October 2016), pp. 145–154. doi: 10.1016/j.cageo.2016.11.004.
- Witten, I. H. et al. (2017) *Data Mining: Practical Machine Learning Tools and Techniques*. Fourth Edi. Elsevier Inc.

Zhang, J. *et al.* (2020) 'Data-Driven Computational Social Science: A Survey', *Big Data Research*, 21, p. 100145. doi: 10.1016/j.bdr.2020.100145.

Zhang, M. *et al.* (2020) 'A database for developing machine learning based disruption predictors', *Fusion Engineering & Design*, 160, p. N.PAG-N.PAG. Available at: <http://10.0.3.248/j.fusengdes.2020.111981>.


Zheng, W. (2020) 'Research trend of large-scale supercomputers and applications from the TOP500 and Gordon Bell Prize', *Science China Information Sciences*, 63(7), pp. 1–14. doi: 10.1007/s11432-020-2861-0.

Zhou, Y. *et al.* (2014) 'Parallel feature selection inspired by group testing', in Z. Ghahramani *et al.* (eds) *Advances in Neural Information Processing Systems 27 (NIPS 2014)*. Curran Associates, Inc. Available at: <https://proceedings.neurips.cc/paper/2014/file/fb8feff253bb6c834deb61ec76baa893-Paper.pdf>.

11. Annex

11.1 Research align with the three Hevner cycles: Relevance, Design and Rigor

Table 11-1 - Research align with Hevner cycles, Relevance, Design and Rigor

Hevner Guidelines (Hevner, 2007)	Peffer's six-stage (Peffer's <i>et al.</i> , 2007)	Work Package decomposition
The relevance of the problem	Problem identification and motivation	Bibliographic research for a clear understanding of the problem and its different contexts, with emphasis on LAID and parallel computing to ensure theoretical foundations.
	Define the objectives for a solution	Systematic Literature Review (SLR) for qualitative finding for possible solutions in analogous or related problems
Design as an artifact	Design and development	Interactive design cycle Arrange of the data on HDF5 logical structure. New development or adaptation of existing algorithms i.e., the construction of the artifact
The design constituting itself as a research process		The experimental component will be carried out using parallel processing in the cloud infrastructure of the National Infrastructure for Distributed Computing (INCD) Outcome evaluation
The evaluation of the design	Demonstration	
Rigor of the Research		Two performance measures will be used to evaluate results: the computational time and the quality of the solutions. The quality of the solution is attained by two criteria: the minimum number of attributes that better explains the dataset and the accuracy of the reduced dataset. (Cavique <i>et al.</i> , 2018)
The research contributions	Evaluation	Final evaluation of the best solution obtained and evaluation of the usefulness of the artifact Also comprises fitting the Design Evaluation Methods of the artifact described by (Hevner <i>et al.</i> , 2004) in three dimensions; Analytical, Experimental and Testing
Communication of the research and its results	Communication	Write and submit a paper to scientific journals

11.2 The Artifact

11.2.1 Configuration file

```
1  [
2  [
3  [
4      "/users/hpc/pmorgado/laidp/data/raw/",
5      "laidp_original_dataset_1000k.h5",
6      2000,
7      1000000,
8      1,
9      2,
10     "CONTIGUOUS",
11     0,
12     0,
13     "T6_1000k",
14     "laidp_dmatrix_",
15     5000,
16     0,
17     "laidp_aux.h5",
18     "Y",
19     "Y",
20     "Y"
21 ]
22 ],
23 [
24     [ "D1C0", "D1C0.txt", 0, 0, 0, 1699, 199999, 0 ],
25     [ "D1C1", "D1C1.txt", 1, 1700, 0, 1999, 199999, 0 ],
26     [ "D2C0", "D2C0.txt", 0, 0, 200000, 1699, 399999, 0 ],
27     [ "D2C1", "D2C1.txt", 1, 1700, 200000, 1999, 399999, 0 ],
28     [ "D3C0", "D3C0.txt", 0, 0, 400000, 1699, 599999, 0 ],
29     [ "D3C1", "D3C1.txt", 1, 1700, 400000, 1999, 599999, 0 ],
30     [ "D4C0", "D4C0.txt", 0, 0, 600000, 1699, 799999, 0 ],
31     [ "D4C1", "D4C1.txt", 1, 1700, 600000, 1999, 799999, 0 ],
32     [ "D5C0", "D5C0.txt", 0, 0, 800000, 1699, 999999, 1 ],
33     [ "D5C1", "D5C1.txt", 1, 1700, 800000, 1999, 999999, 1 ]
34 ]
35 ]
```

Figure 11-1 - config.json

First config block legend:

1. "Path of raw data files",
2. "Source-Dataset file name",
3. "Number of rows",
4. "Number of features",
5. "Number of classes",
6. "Number different values on class",
7. "Chunk type ('CONTIGUOUS', 'CHUNKED')",
8. "Chunk row or zero (0)",
9. "Chunk columns or zero (0)",

10. "Experiment reference",
11. "DM-Dataset file name",
12. "parameter Step for sort",
13. "Number of features used for test only. zero (0) for all",
14. "Auxiliary Dataset File name",
15. "Perform Laid step 1 - Check and fix redundant and/or inconsistent observations) Y/N",
16. "Perform Laid step 2 - Create DM-Dataset Y/N",
17. "Perform Laid step 3 – Find solution Y/N"

Second config block legend:

1. Block
2. File
3. Class
4. Row starts
5. Column starts
6. Row ends
7. Col ends
8. Rightmost column

11.2.2 Python Code

Due to the size of this code and to facilitate its future use, the option followed was to publish it in a public code repository. It can be found at the URL: <https://github.com/ppmorgado/parallel-laid>

11.3 Sample Output

```
* -----
* Running PROLOG for LaidS on Fri Aug 20 04:46:58 WEST 2021
*   JOB_NAME           : LaidS
*   JOB_ID             : 2017710
*   JOB_PARTITION     : hpc
*   JOB_USER          : pmorgado
*   JOB_ACCOUNT       : hpc
*   JOB_QOS           : normal
*   NODE_LIST        : hpc049
*   SLURM_NNODES     : 1
*   SLURM_NPROCS    : 1
*   SLURM_NTASKS    : 1
*   SLURM_TASKS_PER_NODE : 1
*   SLURM_JOB_CPUS_PER_NODE : 1
*   SLURM_MEM_PER_CPU  : 8000
*   WORK_DIR         : /users/hpc/pmorgado/laidp/t6
* -----
Start job: 2017710
2021-08-20 04:46:58,531340839
Currently Loaded Modulefiles:  1) gcc-4.8           2) mvapich2/2.3.5   3) hdf5/1.12.0
python version: Python 3.6.9
working dir: /users/hpc/pmorgado/laidp/t6
disk usage before: 1.9G.
/users/hpc/pmorgado/laidp/t6/config.json
Task_0; Laid Serial 2021 version
Task_0; Experiment T6_1000k; Storage Layout of Sample Dataset CONTIGUOUS
Task_0; Dataset from HDF5 /users/hpc/pmorgado/laidp/t6/laidp_original_dataset_1000k.h5
Task_0; Settings loaded from HDF5 metadata:
Task_0, number_of_rows      : 2000
Task_0; number_of_columns   : 100000
Task_0; number_of_features  : 100000
Task_0; number_jnsq_features : 0
Task_0; number_of_classes   : 1
Task_0, Shape class_array   : (2000, 1)
Task_0; Check and Fix redundant and/or inconsistent observations Y/N: Y
Task_0; Sort columns; 21 number of column blocks to handle
Task_0; Columns Sorted in; 155.8016; seconds
Task_0; ordered_array shape (2000,) and content sample
Task_0; positional_array shape (2000,) and content sample
Task_0; Check redundant and/or inconsistent observations
Task_0; Found (0) Redundant and (0) Inconsistent rows; In; 0.0012; seconds
Task_0; redundant_array shape (0, 1) and content sample
Task_0; inconsistent_array shape (0, 2) and content sample
Task_0; Handle required jnsq attributes
Task_0; created (0) jnsq attributes; In; 0.0000; seconds
Task_0; jnsq_array shape (2000, 0) and content sample
Task_0; shape disjoint_array : (100000,)
Task_0; max of disjoint rows (worst case);1999000 effective disjoint rows;510000
Task_0; Disjoint Matrix dataset: [510000,100000]
Task_0, Parallel step = 2000
Task_0, start = 0 stop = 1999

Task_0; Row : 0; Class: [0]; Number of interact: 1999; Disjoint rows found: 300; in; 0.1899;
seconds
Task_0; Row : 1; Class: [0]; Number of interact: 1998; Disjoint rows found: 300; in; 0.1981;
seconds
Task_0; Row : 2; Class: [0]; Number of interact: 1997; Disjoint rows found: 300; in; 0.1983;
seconds
Task_0; Row : 3; Class: [0]; Number of interact: 1996; Disjoint rows found: 300; in; 0.1916;
seconds
Task_0; Row : 4; Class: [0]; Number of interact: 1995; Disjoint rows found: 300; in; 0.1933;
seconds
```

```

Task_0; Row : 5; Class: [0]; Number of interact: 1994; Disjoint rows found: 300; in; 0.1916;
seconds
Task_0; Row : 6; Class: [0]; Number of interact: 1993; Disjoint rows found: 300; in; 0.1929;
seconds
Task_0; Row : 7; Class: [0]; Number of interact: 1992; Disjoint rows found: 300; in; 0.1937;
seconds
Task_0; Row : 8; Class: [0]; Number of interact: 1991; Disjoint rows found: 300; in; 0.1956;
seconds
Task_0; Row : 9; Class: [0]; Number of interact: 1990; Disjoint rows found: 300; in; 0.1915;
seconds
(...)
Task_0; Row : 1982; Class: [1]; Number of interact: 17; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1983; Class: [1]; Number of interact: 16; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1984; Class: [1]; Number of interact: 15; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1985; Class: [1]; Number of interact: 14; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1986; Class: [1]; Number of interact: 13; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1987; Class: [1]; Number of interact: 12; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1988; Class: [1]; Number of interact: 11; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1989; Class: [1]; Number of interact: 10; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1990; Class: [1]; Number of interact: 9; Disjoint rows found: 0; in; 0.0004;
seconds
Task_0; Row : 1991; Class: [1]; Number of interact: 8; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1992; Class: [1]; Number of interact: 7; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1993; Class: [1]; Number of interact: 6; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1994; Class: [1]; Number of interact: 5; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1995; Class: [1]; Number of interact: 4; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1996; Class: [1]; Number of interact: 3; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1997; Class: [1]; Number of interact: 2; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1998; Class: [1]; Number of interact: 1; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0; Row : 1999; Class: [1]; Number of interact: 0; Disjoint rows found: 0; in; 0.0003;
seconds
Task_0 - Disjoint matrix generated in; 440.4836; seconds
Task_0; max interact (worst case);1999000 effective interact;1999000
Task_0; Total number_of_interact      : 1999000
Task_0; Total rows_on_disjoint_matrix : 510000
Task_0; Estimated file size 47.497451305389404 GB without compression

Task_0; selected_column; 10594
Task_0; selected features; 1; interactions needed #;510000; time spent;1220.1627; seconds
Task_0; selected_column; 68429
Task_0; selected features; 2; interactions needed #;307600; time spent;1477.6641; seconds
Task_0; selected_column; 43739
Task_0; selected features; 3; interactions needed #;181876; time spent;1230.3814; seconds
Task_0; selected_column; 14663
Task_0; selected features; 4; interactions needed #;105620; time spent;974.2748; seconds
Task_0; selected_column; 64308
Task_0; selected features; 5; interactions needed #;60225; time spent;557.5738; seconds
Task_0; selected_column; 22918
Task_0; selected features; 6; interactions needed #;34257; time spent;175.1373; seconds
Task_0; selected_column; 99741
Task_0; selected features; 7; interactions needed #;19238; time spent;14.0505; seconds
Task_0; selected_column; 87276

```

```

Task_0; selected features; 8; interactions needed #;10839; time spent;7.9455; seconds
Task_0; selected_column; 72734
Task_0; selected features; 9; interactions needed #;6179; time spent;4.5066; seconds
Task_0; selected_column; 2428
Task_0; selected features; 10; interactions needed #;3480; time spent;2.5489; seconds
Task_0; selected_column; 98231
Task_0; selected features; 11; interactions needed #;1884; time spent;1.3840; seconds
Task_0; selected_column; 50112
Task_0; selected features; 12; interactions needed #;985; time spent;0.7270; seconds
Task_0; selected_column; 8801
Task_0; selected features; 13; interactions needed #;531; time spent;0.3878; seconds
Task_0; selected_column; 98460
Task_0; selected features; 14; interactions needed #;269; time spent;0.1956; seconds
Task_0; selected_column; 33426
Task_0; selected features; 15; interactions needed #;135; time spent;0.1016; seconds
Task_0; selected_column; 86941
Task_0; selected features; 16; interactions needed #;57; time spent;0.0423; seconds
Task_0; selected_column; 97079
Task_0; selected features; 17; interactions needed #;21; time spent;0.0165; seconds
Task_0; selected_column; 6
Task_0; selected features; 18; interactions needed #;3; time spent;0.0032; seconds
Task_0; selected features; 18; interactions needed #;0; time spent;0.0007; seconds
Task_0; Solution found in; 5667.1110; seconds
Task_0, number_selected_features: 18
Task_0; shape selected_feature_array;(100000,)
Task_0; selected_feature_array;[ 6 2428 8802 10594 14664 22920 33431 43740 50117 64311
68430 72740 86952 87282 97092 98240 98472 99747]

Task_0; LAID concluded
Task_0 Try close all HDF5 Files
Task_0 HDF5 Files Closed
Finished job 2017710
2021-08-20 06:31:23,640926752
disk usage after: 50G

```