Seleção de atributos usando LAID e sua implementação em sistemas de computação de alto desempenho

Paulo Morgado[1], Luis Cavique[2]

[1] Mestre em Informação e Sistemas Empresariais, Instituto Superior Técnico e Universidade Aberta
[2] Universidade Aberta, DCeT
pmorgado.info@gmail.com, luis.cavique@uab.pt

Resumo

Do conjunto de técnicas de redução de dimensionalidade focamo-nos na seleção de atributos uma possível abordagem para a realizar é a utilização da Análise Lógica de Dados Inconsistentes (LAID). Recentemente vários estudos demostraram as suas potencialidades na resolução deste problema e evidenciaram as suas vantagens como uma metodologia robusta, de fácil interpretação e adicionalmente capaz de lidar com dados inconsistentes. Os mesmos estudos revelaram tempos de processamento acima do desejado para uma utilização plena e preconizaram a execução dos algoritmos através de processamento paralelo com recurso a computação de alto desempenho (HPC). Este trabalho representa mais um contributo nesse esforço ao abordar formas de armazenamento dos dados, soluções de paralelização dos algoritmos, configuração do ambiente HPC e finalmente os testes na Infraestrutura Nacional de Computação Distribuída (INCD) que permitiram extrair as conclusões apresentadas.

Palavras-chave: Ciência de dados, data mining, Seleção de atributos, Análise lógica de dados inconsistentes (LAID), Computação paralela

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

Abstract: From the set of dimensionality reduction techniques, we focus on the feature selection, a possible approach to carry it out is the use of Logical Analysis of Inconsistent Data (LAID). Recently, studies have demonstrated its potential in solving this problem and highlighted its advantages as a robust methodology, easy to interpret, and additionally capable of dealing with inconsistent data. The same studies revealed processing times higher than desired for full use and recommended the execution of algorithms through parallel processing using high-performance computing (HPC). This work represents yet another contribution to this effort by addressing ways of storing data, solutions for parallelizing algorithms, configuring the HPC environment, and finally testing the National Infrastructure for Distributed Computing (INCD) that allowed us to draw the conclusions presented.

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

1. Introdução

O nosso modo de vida é cada vez mais impulsionado por dados [Pentland 2013] e dependente de nossa capacidade de coletar, armazenar, processar, analisar e extrair conhecimento de uma quantidade cada vez maior de dados em tempo útil.

O mesmo fenômeno pode ser observado em ambientes científicos [Zhang et al 2020]. O crescimento exponencial de dados gerados por experiências científicos, instrumentos e sensores, considerando seu volume, complexidade e escala de distribuição, tornou-se um fator crítico em várias disciplinas de ciências, o que induz a necessidade de analisar volumes cada vez maiores de dados em tempo útil. Alguns autores identificaram essa necessidade como um novo paradigma na produção científica e adotaram o termo cunhado por Jim Gray, "O Quarto Paradigma" [Hey, Tansley, Tolle 2009]. Segundo eles, a ciência começou empírica, depois teórica, nas últimas décadas tornou-se computacional e, recentemente, passou a ser orientada por dados. Outro autor [Agrawal, Choudhary 2016] chama essa tendência de "ciência intensiva em dados", que consiste em três atividades essenciais: captura, curadoria e análise.

Seja na esfera científica, empresarial ou governamental, a mesma necessidade levou à criação de novas formas de gerir dados distribuídos, processando-os e, principalmente, analisando-os para obter informações e conhecimentos úteis.

Este processamento e análise são habitualmente denominados como descoberta de conhecimento a partir de dados ou KDD de *knowledge discovery from data*. Deste processo uma etapa essencial [Azevedo, Santos 2008] é conhecida como Data mining que pode ser definida como o processo de descobrir padrões e extrair conhecimento de grandes quantidades de dados [Han, Kamber, Pei 2012]. Padrões em dados é uma expressão genérica que agrega conceitos como identificar agrupamentos de dados não conhecidos anteriormente, detetar registos incomuns ou encontrar dependências desconhecidas. Esse processo pode ser automático ou, muitas vezes, semiautomático [Witten et al. 2017].

Dado ser normalmente inviável analisar todo o conjunto de dados, (*Dataset)*, é preciso saber escolher em quais atributos a análise se deve focar, em detrimento de todas as outras. É nesse âmbito que o presente trabalho se concentrou, especificamente nas situações em que a seleção de atributos ou *features* torna-se imprescindível, como no caso de conjuntos de dados com elevado número de dimensões, como os habitualmente denominados *Omics Datasets*. Omics é um neologismo associado a sufixos biológicos, como genómica, transcriptómica ou proteómica, [Cavique et al. 2018] e geralmente refere-se a parâmetros biológicos.

Nesse sentido, a redução de dimensões é a especialidade no âmbito do habitualmente denominado *data mining*, e uma das formas possíveis de realizá-la é através da seleção de atributos, obrigatória em conjuntos de dados altamente dimensionados para os quais a análise de todas as dimensões é não só impraticável como contraproducente. Esta tarefa torna-se ainda mais necessária quando o número de observações é baixo, a matriz

de dados é esparsa e a análise da classificação das linhas revela inconsistências. O problema abordado neste trabalho é um exemplo disso. Considerando que um problema de seleção de atributos com um *dataset* envolvendo um milhão de atributos foi resolvido em [Cavique et al. 2018], usando partições do problema e um novo método denominado Análise Lógica de Dados Inconsistentes, LAID de *Logical Analysis of Inconsistent Data* proposto por [Cavique, Mendes, Funk 2011], a presente pesquisa tem como objetivo continuar este esforço.

A experiência de 2018 [Cavique et al. 2018] utilizou um cluster de computadores da Infraestrutura Nacional de Computação Distribuída (INCD) de Portugal, para processamento paralelo, utilizando o *dataset* em memória. Como comentário final, os autores consideraram três aspetos principais: os dados, o algoritmo e ambiente de computacional. Uma vez que grandes *datasets* com muitas dimensões tendem a tornar-se o novo normal, eles defendem que:

- O acesso aos dados na memória deve ser substituído pelo acesso aos dados em disco.
- Os novos algoritmos precisam de menor complexidade em termos de tempo e apoiar-se na decomposição do problema por forma a permitir a sua paralelização. Nesse sentido, o uso intensivo de algoritmos computacionalmente dispendiosos, como por exemplo meta heurísticas, deve ser reduzido.
- E por último, o ambiente de computação ideal deve ser na CLOUD, executando os programas em paralelo com recurso a computação de alto desempenho (HPC) na sigla inglesa.
-

O que está de acordo com as novas tendências para algoritmos de computação intensiva [Talia 2019] e constitui mesmo uma mudança de paradigma em ambientes científicos, [Cavique et al. 2018], ilustrado na Figura 1.

	dados	algoritmos	ambiente
antes	em memória	sequencial	local
agora	em disco	paralelo	cloud

Figura 1 – Mudança de paradigma nos ambientes científicos

Em 2019 [Apolónia, Cavique 2019a] revisitou o problema novamente usando LAID, mas uma abordagem computacional diferente: recorrendo a processamento sequencial em uma única máquina e o dataset armazenado em disco no *Hierarchical Data Format* Versão 5 (HDF5).

Ambos as experiências concluíram que o tempo de processamento constitui um problema adicional que limita a sua aplicação. Por esse motivo, o desafio e a motivação para esta nova abordagem ao problema é reduzir o tempo de processamento, utilizando o novo paradigma computacional: processamento paralelo, acesso a dados em disco e ambiente HPC na Cloud.

O objetivo deste trabalho é descobrir como o processamento paralelo pode reduzir os tempos de processamento para o utilizador final e descrever como este novo paradigma computacional se aplica a este problema.

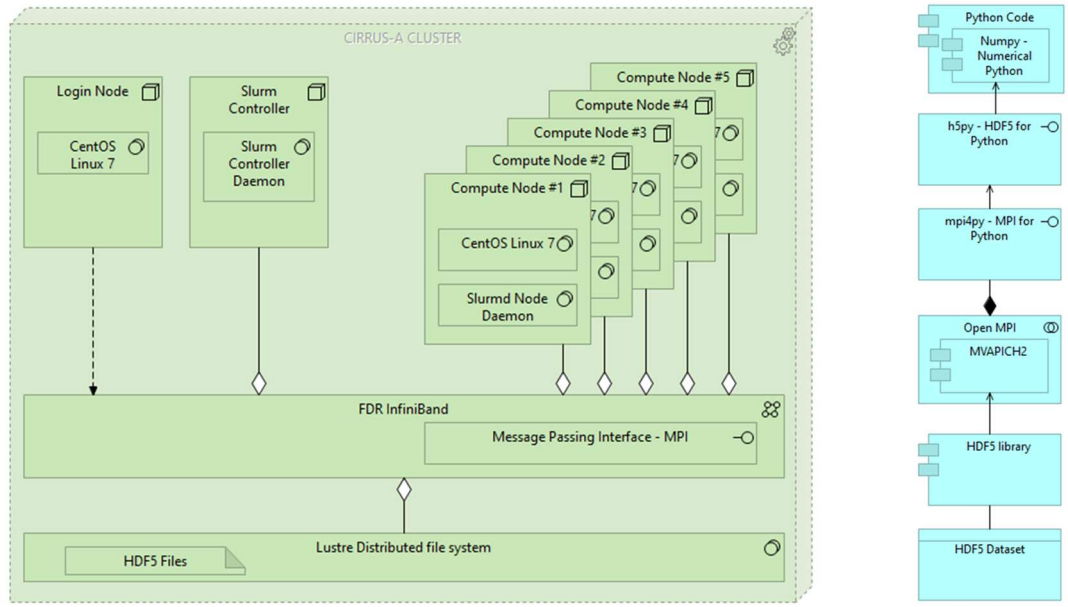
2. O ambiente HPC na Cloud do INCD

HPC, também conhecido como supercomputação, refere-se a sistemas de computação com elevado poder computacional que podem resolver problemas extremamente complexos e exigentes [Ec.europa.eu 2021] e [Zheng 2020], o desempenho é habitualmente medido em número de operações de virgula flutuante por segundo, FLOPS na sigla inglesa.

O INCD é uma infraestrutura digital de apoio à investigação, prestação de serviços de informática e armazenamento à comunidade científica e académica portuguesa. Gentilmente disponibilizou recursos computacionais no cluster HPC Cirrus-A, que permitiram a execução e teste do código que constitui o artefacto deste trabalho. Neste cluster, existem 5 nós de computação AMD EPYC 7501, que correm o CentOS 7, com 16 núcleos por nó computacional, interconectados por rede de baixa latência FDR InfiniBand 56Gbps [INCD n.d.). O sistema de ficheiros distribuídos do INCD é baseado em Lustre *distributed filesystem*.

A figura 2a) apresenta esquematicamente o *cluster* Cirrus-A e os principais componentes utilizados. Destaque para os componentes essenciais para este trabalho, como HDF5 modelo de dados, biblioteca e formato de arquivo para armazenamento e gestão de dados. [The HDF Group n.d.] e a Message-Passing Interface (MPI), [MPI-Forum n.d.], que endereça o modelo de programação paralela com recurso a troca de mensagens e MPI-I/O que fornece rotinas para manipulação de arquivos e acesso a dados por vários processos. Figure 2b) apresenta um diagrama de cooperação da aplicação contendo a solução completa de componentes visando a sua paralelização. Parte significativa da escolha dos módulos de software e das versões utilizadas resultou do excelente suporte e conhecimento da equipe que gere e mantem a infraestrutura.

Por se tratar de um ambiente partilhado e competitivo, é essencial conhecer e seguir as regras estabelecidas, para não interferir no trabalho de outros utilizadores e colaborar no objetivo do uso mais eficiente dos recursos partilhados. por exemplo, é necessários um cuidado especial no agendamento de jobs e os recursos necessários, porque pedir muitos recursos de uma vez quando muitos outros utilizadores estão igualmente em fila de espera simplesmente leva a muito tempo de espera até que os recursos pedidos estejam disponíveis. Frequentemente, é preferível solicitar menos recursos e conseguir realizar as tarefas com mais rapidez.



a) Ponto de vista do uso da tecnologia do Cluster HPC Cirrus-A

b) Ponto de vista cooperação de aplicações

Figura 2 - *cluster* Cirrus-A, pontos de vista de tecnologia e cooperação de componentes

3. O *dataset* utilizado e a estratégia de armazenamento no formato HDF5

3.1. O dataset utilizado

O *dataset* utilizado (dataset-Origem) usado contém um total de 2k observações x 1000k atributos de dados sintéticos que representam algum tipo de dados Omic, todas as observações estão rotuladas de acordo com uma classificação binária. Foi recebido na forma de arquivos de texto e foi desenvolvido um programa em Python para extrair dados de ficheiros de texto e carregá-los no ficheiro HDF5, com recurso a computação paralela.

Após o seu carregamento no formato HDF5, foi realizada uma breve análise exploratória de dados, que permitiu concluir que apenas 19,57% das células contêm um valor, o que significa uma matriz esparsa. Permitiu igualmente concluir que a distribuição não é uniforme ao longo das colunas, de facto existem dois diferentes padrões de distribuição intercalados entre blocos de 100k colunas onde um padrão de distribuição aparentemente aleatório. Figura 3 a) alterna com um padrão mais organizado e concentrado. Figura 3 b), sendo que neste último, 31% das colunas não contém qualquer valor.

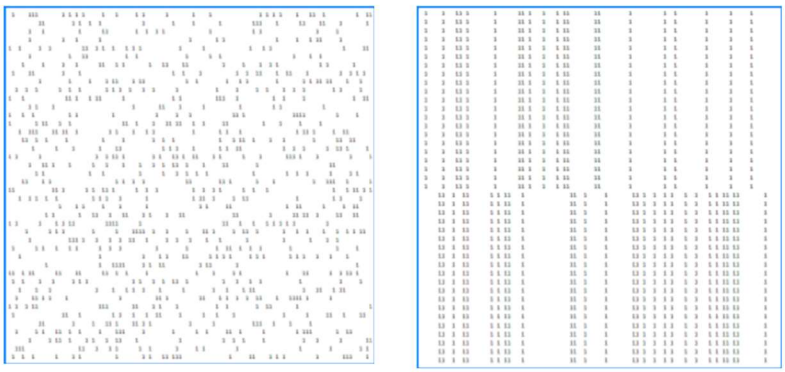


Figura 3 - Exemplos de padrões encontrados no Dataset-Origem

3.2. Layout de armazenamento HDF5: Análise de compartimentação (Chunking)

O layout de armazenamento padrão de arquivos HDF5 é o armazenamento contíguo, "os dados de uma matriz multidimensional são serializados ao longo da dimensão de mudança mais rápida e são armazenados como um bloco contíguo no arquivo. Este mecanismo de armazenamento é recomendado se o tamanho de um conjunto de dados for conhecido e o tamanho do armazenamento para todo o conjunto for aceitável para o utilizador" [Andrew Collette and contributors n.d.].

Os *datasets* também podem ser criados usando o layout de armazenamento em blocos ou compartimentado (*Chunked*) do HDF5. Isso significa que o conjunto de dados é dividido em pedaços de tamanho regular que são armazenados aleatoriamente no disco e indexados usando uma árvore binária (*B-tree*). Esse armazenamento possibilita igualmente redimensionar *datasets*, após o armazenamento e como os dados são armazenados em blocos de tamanho fixo, permite ainda a utilização de filtros de compressão.

Por esses motivos, a escolha do layout de armazenamento e da organização interna do arquivo HDF5 requer análise e testes que suportem uma decisão informada. Para realizá-lo, foi definido um protocolo de teste considerando três subconjuntos do dataset-Origem, com base no número de colunas: L, XL e XXL descritos na Tabela 1.
As diferentes opções testadas são descritas na Tabela 2. Uma explicação adicional é necessária quanto ao *Best-Fit-Chunk* ou bloco que melhor se ajusta ao tamanho da *cache* de leitura da instalação HDF5, depende do conhecimento prévio desse parâmetro que pode ser obtido com recurso às funções da API de baixo nível h5py [Andrew Collette and contributors n.d.]. O valor obtido foi 1048576 bytes.

Tabela 1 - *Datasets* usados no protocolo de teste de leitura / escrita

Dataset	Linhas x colunas	GB
200K (L)	2k x 200k	0,37
600K (XL)	2k x 600k	1,12
1000K (XXL)	2k x 1000k	1,86

Tabela 2 - Layout de *chunking* testado

Layout	Significado
Row-Chunk	Equivale à dimensão de uma linha, independentemente da dimensão desta
Best-Fit-Chunk	a dimensão ajusta-se ao tamanho definido da *cache* de leitura do HDF5
Contiguous	Sem compartimentação
Auto-Chunk	Atribui ao HDF5 a responsabilidade pela escolha da dimensão

3.3. Operações de escrita sobre dataset-Origem em HDF5

Esses testes foram realizados com recurso a sucessivas operações de escrita paralela sobre o dataset-Origem e todos os tempos apresentados são a média das execuções realizadas. Digno de destaque que do tempo total de carregamento dos dados, cerca de 60% do tempo é consumido a extrair os dados do arquivo de texto, 40% na sua transformação em memória e menos de 0,5% é gasto na preservação dos dados no ficheiro HDF5. Isso é conseguido em uma única operação de gravação que provou ser extremamente eficiente, independentemente do tamanho do bloco de dados e do layout de armazenamento escolhido.

Verificou-se nestes testes uma excessiva variabilidade dos tempos entre execuções semelhantes. as cargas de trabalho e as condições de I/O do cluster notoriamente influenciaram os tempos obtidos principalmente nos testes L, porém, e dado que na parcela referente à operação de escrita apresentada na Tabela 3, são tempos muito baixos, (todos abaixo de 8 segundos), para extrairmos conclusões o foco tem de se colocar em pequenas diferenças (na ordem dos segundos) e aqui dois dos layouts destacaram-se pela negativa; *Row-Chunk* para *datasets* com menos de 1000K colunas e *Auto-Chunk* cujo desempenho foi tão baixo que foi abandonado após os primeiros testes.

Tabela 3 - Tempo médio para gravar um bloco de dados no arquivo HDF5 em segundos

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		

Com relação ao recurso a *chunking* nas operações de escrita e exceto para *Auto-Chunk*, não há grandes discrepâncias nos tempos observados, considerando o tamanho do conjunto de dados e o *chunk* usado. Não usar compartimentar de todo não é penalizado nas operações de escrita e pode até ser mais rápido em *datasets* com 600 mil colunas. A opção *Best Fit-Chunk* é o mais complexo de utilizar e provavelmente a melhor solução para alguns tamanhos, onde exista melhor ajuste entre a dimensão da linha e o tamanho do cache HDF5.

3.4. Operações de leitura sobre o Dataset-origem em HDF5

Neste contexto, o layout de armazenamento deve estar alinhado com os padrões de acesso aos dados, ou seja, as operações de leitura, que nesse sentido a sua análise é mais relevante do que a análise das operações de escrita. Daí a escolha de realizar estes testes utilizando o código da 2ª etapa da metodologia LAID, nesta etapa o acesso ao Dataset-Origem requer a leitura completa da linha, pois cada linha é comparada com todas as outras, portanto, esta leitura padrão é perfeitamente adequada para o teste pretendido. O gráfico da Figura 4 compara o tempo despendido na leitura nas diferentes opções de layout de armazenamento HDF5.

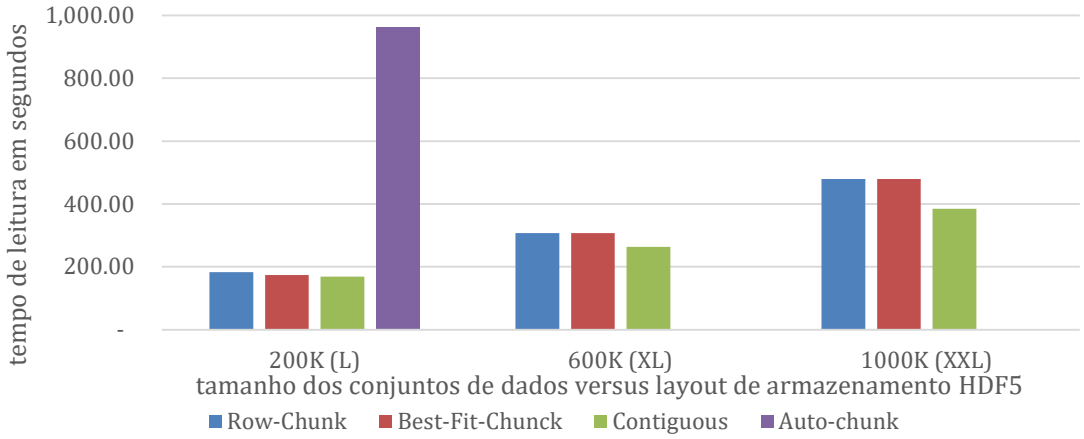


Figura 4 - Tempo médio em segundos das operações de leitura versus layout de armazenamento HDF5

A opção *Best Fit-chunk* praticamente corresponde à dimensão em bytes da linha XXL, também é possível fazer uma correspondência razoável com o L usando 5 linhas por *chunk*. Mas o XL é penalizado porque desperdiça quase metade do valor do cache em cada leitura. Mesmo assim, e exceto para a opção *Auto-chunk*, que tem um baixo desempenho mesmo em pequenos *datasets* que levou a não ser sequer testado com os maiores, os tempos entre as duas opções de *chunking* restantes são relativamente semelhantes.

Encontrou-se uma relação linear entre o tempo de leitura e o número de colunas no *dataset*, mais evidente na opção contígua.

3.5. Síntese

Ficou confirmado que o formato HDF5 é uma excelente opção para armazenar os *datasets* necessários a uma análise LAID. É uma boa prática preservar o *dataset* original ordenado pelos valores de classe porque facilita as fases subsequentes do processo e uma vez registados em ficheiro a sua reordenação consumiria tempo precioso.
Embora não tenha implicações de desempenho, separar o *dataset* original e os derivados em diferentes ficheiros HDF5 revela-se uma boa opção, pois permite uma melhor gestão dos mesmos, como cópia, compactação, exclusão, etc.

A escolha do tipo de dados para as informações, bem como o layout de armazenamento, deve ser feita com cuidado ao invés de recorrer à parametrização predefinida do HDF5.
Dos testes efetuados sobre o padrão de leitura verifica-se que mesmo para a melhor opção de *chunking* considerando o tamanho do cache de leitura definido na instalação do HDF5 e o padrão de leitura utilizado pelo algoritmo, observa-se que o layout de armazenamento contíguo permite sistematicamente maior velocidade de acesso. Esta velocidade tem pouco significado em conjuntos de dados menores (L), mas gradualmente torna-se mais significativa. Do que se pode concluir que o recurso a *chunking* é uma opção, mas de forma alguma deve ser considerada uma obrigação e a estratégia de layout preferida para o Dataset-Origem é não usar *chunking*.

Adicionalmente para o DM-Dataset obtido na 2ª etapa do LAID, por o seu padrão de leitura ser o acesso em linha e coluna, utilizar qualquer forma de *chunking* é um erro.

4. LAID em paralelo

4.1. Metodologia LAID

A metodologia LAID surge da combinação da Teoria dos *Rough Sets* [Pawlak 1991] e da metodologia de Análise Lógica de Dados (LAD) [Boros et al. 1997]. Ambas as abordagens são um subconjunto de modelos de filtro, cujo objetivo é reduzir o número de atributos do *dataset* usando duas fases: transformação e otimização do problema. A sua maior especificidade é manter a semântica dos dados removendo apenas os considerados redundantes.

A LAID foi proposta em 2011 por [Cavique, Mendes, Funk 2011] no âmbito de um Estudo Paremiológico e é descrita pelos autores como uma mistura das melhores qualidades de cada uma das metodologias anteriores, sendo capaz de lidar com dados inconsistentes e classes não dicotomizadas, características de *Rough Sets*, [Rissino, Lambert-Torres 2009] bem como a eficácia computacional da LAD. Alem disso permite atributos inteiros, com custos associados. [Cavique et al. 2013] o que amplia a variedade de estratégias de seleção. Documentação detalhada com exemplos pode ser encontrada em [Apolónia, Cavique 2019b].

4.2. Metodologia de design para programação paralela

A metodologia de Foster, descrita por [Pacheco 2011], indica uma sequência de etapas que podem ser usadas no desenvolvimento de programas paralelos ou na adaptação de código sequencial. As etapas são:

- Partição - O processo de divisão de dados e computação em "peças" menores,
- Comunicação - O processo de definir o que e como as tarefas se comunicam entre si,
- Aglomeração ou agregação - O processo de agrupar tarefas em tarefas maiores para simplificar e melhorar o desempenho,
- Mapeamento - O processo para atribuir tarefas a processos / processadores

O objetivo da partição é descobrir o máximo de paralelismo possível, foi amplamente usado neste trabalho conforme descrito nas páginas seguintes.

A aglomeração também foi usada pela concentração de todo o código sequencial em um único bloco de código para ser executado de uma vez.

Mapeamento é o procedimento de atribuir cada tarefa a um processador, foi intensamente utilizado neste trabalho, considerando que o objetivo no desenvolvimento de algoritmos de mapeamento é minimizar o tempo total de execução, que pode ser obtido minimizando a comunicação entre processadores (ou eliminando-a de todo) e maximizando o uso de cada processador. Também foi utilizado associado à partição de dados.

Pelo contrário, a comunicação não foi usada de todo. Aqui é importante mencionar que o uso de memória distribuída, como modelo de programação, requer o uso de APIs paralelas, e é por isso que o MPI [Talia 2019] tem um papel central neste trabalho. Devido ao tempo disponível para o concluir, bem como a intenção de manter uma base de código comum para as versões serie e paralelas, a opção seguida foi não usar o modo coletivo do MPI, antes recorrendo ao funcionamento independente do MPI. Desta forma cada uma das tarefas funciona de forma totalmente independente, sem comunicação e coordenação coletiva ou ponto a ponto, entre os nós computacionais.

4.3. Codificação dos algoritmos LAID

Dada a disponibilidade prévia de uma base de código para os algoritmos LAID nas linguagens de programação C e Python, desenvolvidos em trabalhos anteriores [Cavique et al. 2018] e [Apolónia, Cavique 2019a], esta base de código foi utilizada neste trabalho, com ênfase em a versão Python que já usava HDF5, dessa forma parte do trabalho consistiu em adaptar o código sequencial e os restantes desenvolvimentos necessários, como por exemplo melhorias de código, mudanças no fluxo de execução, introdução de um ficheiro de configuração, uso diferente de ficheiros e *datasets* HDFS e o mais importante a introdução de técnicas visando a sua paralelização.

4.3.1. Etapas LAID

De acordo com [Cavique et al. 2018] e [Apolónia, Cavique 2019a], a metodologia LAID consiste nas seguintes etapas:
Input: *dataset* D = {O, X∪C} com variáveis binárias
Output: (número de atributos, precisão)
1. verifica as inconsistências de dados e adiciona variáveis fictícias "jnsq" como um recurso discriminante para remover qualquer inconsistência, adicionalmente verifica e remove todas as observações redundantes.
2. geração da matriz disjunta [Ai,j]
3. número de atributos = Problema de cobertura do conjunto mínimo.
4. precisão = Validação cruzada

Mesmo sem paralelização a etapa 1 é de execução rápida, por outro lado os algoritmos são complexos de paralelizar, por esse motivo não foi abordada neste trabalho. Por sua vez a etapa 4 consiste em validar a solução obtida, e não requer paralelização.

4.3.2. Etapa 2 - Geração de Matriz Disjunta

Esta etapa consiste em ler todas as linhas do Dataset-Origem e compará-las com todas aquelas que possuem um valor de classe diferente, as linhas disjuntas encontradas são gravadas em um *dataset* derivado, que por razões operacionais se decidiu armazenar em um ficheiro HDF5 separado, a este novo *dataset* derivado foi atribuído o nome de DM-Dataset.

Como não existe dependência entre cada iteração do algoritmo usado, ou seja, cada linha do Dataset-Origem é tratada de forma independente, a sua paralelização parece apenas necessitar de pequenas adaptações ao código sequencial. Tecnicamente é apenas necessário distribuir as linhas do Dataset-Origem pelos processadores utilizados, considerando uma partição de dados horizontal.

Na prática revelou-se mais complexo. Todas as tarefas paralelas gravam as linhas geradas no mesmo *dataset* HDF5. Se esse processo consistisse em adicionar novas linhas ao *dataset*, o processo seria simples, mas na realidade, os *datasets* HDF5 usados são de dimensão fixa, definida no momento da criação. Isso implica para evitar sobrescrever dados, controlar e segregar o índice de linha em cada gravação.

É possível definir uma fórmula para calcular o índice de linhas a gravar, mas somente se o Dataset-Origem se encontrar ordenado pelo valor da(s) classe(s), sucede que essa ordenação apesar de desejável não é um requisito da LAID logo não pode ser considerado um pressuposto e sem ele o problema parece insolúvel, pois antes da execução do algoritmo não é possível conhecer quantas linhas são geradas e quais os índices de linhas onde elas deverão ser escritas. Uma possível solução que foi seguida é executar duas vezes o algoritmo. Na primeira vez, são identificadas as linhas origem e destino, sem despender tempo na sua preservação em disco. Com essas informações da primeira passagem, a segunda conhece exatamente o que e onde escrever. A aposta é que a paralelização permite não só recuperar o tempo gasto na primeira passagem e

ainda poupar tempo, pois evita comparações desnecessárias que não geram linhas disjuntas.

O código foi ainda alterado para incluir um *buffer*, o que permitiu que o número de operações de escrita em disco fosse reduzido escrevendo mais dados em cada operação. A introdução deste revelou-se tão útil que mesmo a versão sequencial beneficia dele, (29% do tempo inicial), o que confirma o axioma de que a melhor maneira de melhorar o desempenho paralelo passa por melhorar o desempenho sequencial.

4.3.3. Etapa 3 - Encontrar a solução com o problema de cobertura mínima de conjuntos

Esta etapa pelo tempo que demora constitui o "gargalo" de todas as etapas da metodologia LAID. O algoritmo desta etapa é inerentemente sequencial dado que existe uma evidente dependência entre as suas interações.

A abordagem seguida foi realizar a partição vertical das colunas do DM-Dataset e sua distribuição por tarefas paralelas. Mas as diferentes soluções encontradas devem ser comparadas e escolhida a melhor antes da interação seguinte. O que requer comunicação e cooperação entre processadores ou tarefas paralelas, portanto, neste trabalho, a abordagem para uma solução paralela foi limitada à primeira iteração. Por esse motivo, a meta de paralelizar essa etapa foi suspensa.

A literatura revela várias outras soluções paralelas potenciais [Blelloch, Simhadri, Tangwongsan 2012] e [Chakravarty, Shekhawat 1992]. Há um ponto comum todos eles necessitam de alguma forma de comunicação entre os processadores utilizados. Em conclusão, a próxima abordagem terá que fazer uso de MPI ponto a ponto ou modos de comunicação coletiva via memória partilhada ou memória externa que permite a coordenação entre tarefas / processadores paralelos.

4.4. Métricas de desempenho para sistemas paralelos

Considerando que o principal objetivo da computação paralela é aumentar a velocidade e reduzir a complexidade do tempo é necessário medir o seu resultado, para tanto, são habitualmente utilizadas duas métricas de desempenho [Cavique et al. 2018] e [Rastogi, Zaheer 2016], para medir a eficiência de um algoritmo em termos do fator de complexidade do tempo. N computadores a operar simultaneamente podem aumentar a velocidade em até N vezes, ou uma aproximação de N, dado que outros fatores podem influenciar o tempo gasto, como estrangulamentos de comunicação.

O tempo de execução paralelo é geralmente definido como o tempo que decorre desde o momento em que um cálculo paralelo começa até o momento em que o último processador termina a execução.

O fator Speedup (Sp) é definido como a razão entre o tempo de execução do melhor algoritmo sequencial para resolver um problema e o tempo gasto pelo algoritmo paralelo para resolver o mesmo problema usando p processadores, expresso na fórmula 1, onde Ts é o tempo de execução sequencial utilizando um único processador. Tp é o

tempo de execução paralelo (com vários processadores) e Sp é o aumento na velocidade usando múltiplos processadores.

$$S_p = \frac{t_s}{t_p} \quad (1)$$

A eficiência E é definida como a razão entre o Sp e o número de processadores P, esta mede a fração de tempo em que um processador é utilizado de maneira útil expresso pela fórmula 2:

$$E = \frac{S_p}{P} \quad (2)$$

5. Experiência computacional e resultados

Todo o código Python foi dotado com contadores de desempenho para benchmarking, cujas informações foram cruzadas com a data e hora de início e término de cada job. Existe, portanto, elevada confiança nos tempos recolhidos, porém, algum cuidado é necessário na sua análise, como referido acima, em várias ocasiões, o mesmo código e os mesmos dados em execução em períodos diferentes retornam o mesmo resultado, mas com discrepâncias no tempo gasto, o que indica que o ambiente de trabalho não é totalmente isolado e é afetado pelas condições operacionais do cluster.

A Figura 5 sintetiza as três formas de execuções utilizadas neste trabalho. O modelo mais simples, representado no lado esquerdo da figura, segue a execução sequencial. A parte central representa a abordagem híbrida, onde parte do processamento é sequencial e parte paralelizado. Finalmente o modelo paralelo é representado do lado direito.

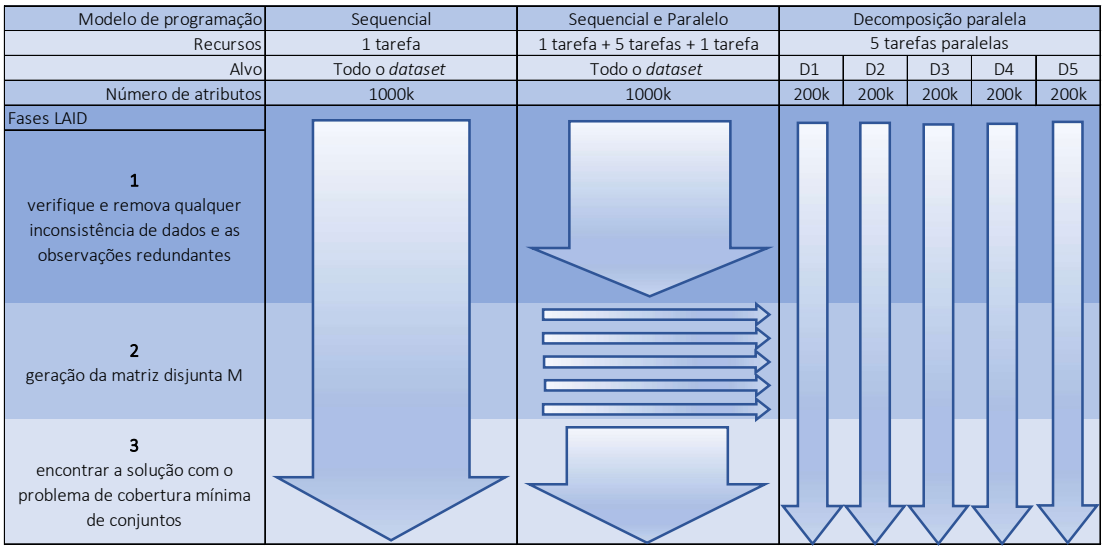


Figura 5 – Abordagens: Sequencial, Híbrida e Decomposição paralela

5.1. Abordagem sequencial

Tabela 4 - Tempo médio de processamento da versão sequencial

Etapa LAID		T0
1 Verifica e remove observações inconsistentes e redundantes		1.254,96
2 Geração da matriz disjunta M	1ª passagem	N/A
	2ª passagem	1.352,21
3 Encontrar a solução com o problema de cobertura mínima de conjuntos		6.221,88
	total em segundos	8.829,06
	total em minutos	147,15

Esta solução é em grande parte uma repetição do procedimento de 2019 [Apolónia, Cavique 2019a] e serve principalmente como uma referência para as atuais experiências de paralelização. O tempo consumido por cada etapa é descrito na Tabela 4 representa aproximadamente metade do tempo obtido em 2019, no entanto, existem demasiadas diferenças entre as duas experiências, os dados, o ambiente HPC, para permitir tirar conclusões relativas à sua comparação.

5.2. Abordagem híbrida com recurso a execução sequencial e paralela

Nesta abordagem, apenas a etapa 2, geração do DM-Dataset, foi paralelizada recorrendo a partição horizontal de dados, assim, as 1700 observações foram divididas por 5 processadores cada encarregue de 340x300 comparações com os resultados descritos na Figura 6.

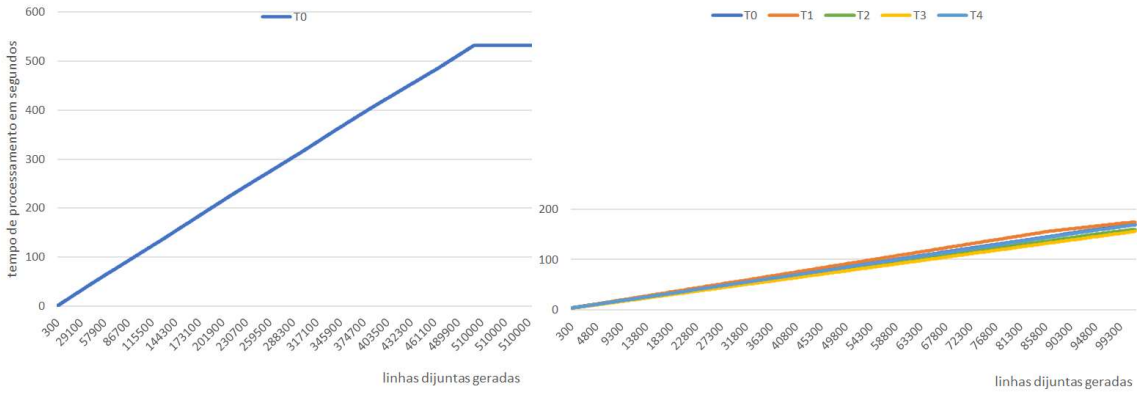


Figura 6 - Geração do DM-Dataset, comparação da versão sequencial versus 5 tarefas paralelas (melhor tempo obtido em ambas as abordagens)

A diferença face à versão sequencial é apreciável, com Sp de 3,04 e eficiência de 0,61 representando apenas 8% do tempo da execução sequencial.

5.3. Abordagem de decomposição paralela com recurso a partição vertical dos dados

A solução anterior, embora mais rápida do que a solução sequencial, não é prática. Considerando as diversas etapas da LAID e não dispondo da paralelização da etapa 3 a solução mais adequada foi optar por paralelizar a execução da versão sequencial, ou seja, correr vários *jobs* em paralelo do código serie em que cada um deles processa subconjuntos do Dataset-Origem no caso, blocos de 200k colunas.

Após a conclusão do processamento, a solução é constituída pelo conjunto das soluções encontradas, podendo ainda ser efetuada uma nova análise sobre este conjunto para obter uma maior redução.

Presume-se com esta solução, o tempo da etapa 3, tenha menor impacto no tempo final. O resultado previsto confirmou-se, comparando os tempos de execução das etapas para 1000k de atributos recorrendo ao processamento sequencial com esta versão paralela, verifica-se que o primeiro demorou 147,15 minutos contra 25,93 minutos do segundo, (Tempo da Tarefa T3 com o pior, por mais demorado, desempenho), o que representa um Speedup de 5,67 para uma eficiência de 1,13.

Tabela 5 - Tempo médio de processamento da versão com decomposição paralela

Etapa LAID			T0	T1	T2	T3	T4
1	Verifica e remove observações inconsistentes e redundantes		32,28	32,60	32,81	33,11	32,98
2	Geração da matriz disjunta M	1ª passagem	211,71	212,91	206,97	211,94	212,05
		2ª passagem	237,44	240,05	236,69	242,38	237,51
3	Encontrar a solução com o problema de cobertura mínima de conjuntos		1.044,77	988,40	956,70	1.068,36	918,17
	total em segundos		1.526,19	1.473,95	1.433,17	1.555,79	1.400,71
	total em minutos		25,44	24,57	23,89	**25,93**	23,35

5.4. Síntese

A Tabela 6 apresenta um quadro comparativo das três abordagens acima descritas, é necessário algum cuidado na comparação etapa a etapa destes resultados, pois o paralelo T3 não pode ser comparado diretamente com as outras abordagens, dado apenas processar 200k colunas. Apenas o tempo total pode ser comparado sem restrições e como esperado, a abordagem totalmente paralela é muito mais rápida do que as outras.

Tabela 6 – Quadro comparativo das 3 abordagens

Etapa LAID			Sequencial	Híbrido	Paralelo T3
1	Verifica e remove observações inconsistentes e redundantes		1.254,96	1.254,96	33,11
2	Geração da matriz disjunta M	1ª passagem	N/A	407,83	211,94
		2ª passagem	1.352,21	444,43	242,38
3	Encontrar a solução com o problema de cobertura mínima de conjuntos		6.221,88	6.221,88	1.068,36
		total em segundos	8.829,06	8.329,11	1.555,79
		total em minutos	147,15	138,82	**25,93**

O resultado da solução considerando todas as colunas do dataset-Origem resume-se à coluna com índice 100004. A razão desta seleção efetuada pelo algoritmo reside no facto de os valores contidos nesta coluna serem exatamente iguais aos valores da coluna da classe, consequentemente a avaliação da qualidade da solução revelaria 100% de precisão e, por esse motivo optou-se por não efetuar a etapa de avaliação da solução obtida.
Existem também duas notas sobre o processamento utilizando decomposição paralela:

1. A adaptação do código sequencial deveria ser simples, uma vez que é efetuada a partição das colunas do Dataset-Origem, é necessário garantir que as linhas geradas a partir do DM-Dataset sejam isoladas para serem processadas por diferentes tarefas paralelas. Para isso, foi tentada a criação de 5 *datasets* no mesmo arquivo HDF5, mas tal não foi possível devido a erros no Lustre no momento da sua criação. O mesmo aconteceu ao tentar a operação em ficheiros HDF5 separados. Em nenhum caso a operação de criação foi possível quando tentada em código executado em paralelo. Devido ao número de componentes e camadas envolvidas, a tentativa de resolução de problemas usando sites especializados como o stackoverflow.com não permitiu identificar a causa e resolver o problema, na lista de causas potenciais estão os componentes principais: h5py, HDF5, a implementação MPI e até mesmo o Lustre. Para contornar o problema, optou-se por criar com antecedência os ficheiros HDF5 usando o modo sequencial e deixar o trabalho de preenchê-los para as tarefas paralelas. Embora essa solução não seja totalmente satisfatória, os resultados finais são bons e testes adicionais serão necessários para resolver definitivamente o problema.

2. Depois de testar 5 tarefas paralelas com sucesso e obter tempos muito aceitáveis, surge naturalmente a questão por que não os dividir em lotes ainda menores e, presumivelmente, obter tempos de execução ainda mais favoráveis? Porém, com este Dataset-Origem essa tentativa revelou-se impossível, como mencionado em 3.1 cada bloco de 200k colunas não é uniforme, e quando tratado separadamente, a segunda metade apresenta um grande número de observações redundantes (1998 em 2000), o que introduziria erros na solução final. Em termos práticos este *dataset* não permite subdivisões abaixo de 200k colunas.

6. Conclusões

Esta investigação faz parte de uma sequência de trabalhos anteriores, [Cavique et al. 2018] e [Apolónia, Cavique 2019a], pretendendo contribuir para a utilização da metodologia LAID no problema de seleção de atributos em *datasets* de grande número de dimensões. Esta contribuição que metaforicamente se constitui como mais um "tijolo" na construção mais ampla do conhecimento visou a solução efetiva de um problema atual e cada vez mais relevante.

A abordagem seguida está em linha com as tendências atuais para algoritmos de computação intensiva, no que é cada vez mais encarado como uma mudança de paradigma em ambientes científicos, onde os dados de trabalho são armazenados em disco, no caso através do formato HDF5 e executando os algoritmos em paralelo em um ambiente de computação de alto desempenho na Cloud.

O objetivo de descobrir como o processamento paralelo pode reduzir os tempos de processamento para o utilizador final e descrever como o paradigma HPC se aplica a este problema levou à decomposição do problema em várias partes a que se procurou dar resposta.
As principais partes constituintes do ambiente de HPC usado foram descritas e documentadas para uso futuro.

Além disso, sobre os dados de amostra utilizado, foram extraídas informações que se espera revelem-se úteis em situações futuras a que se acrescenta um trabalho de análise e comparação das melhores estratégias e layouts para armazenamento da informação em HDF5, atendendo aos padrões de leitura exigidos pelas diferentes etapas do LAID.

O layout contínuo foi indicado como o mais adequado para os conjuntos de dados utilizados e as condições onde as alternativas podem ser utilizadas foram descritas.

Ainda relativamente ao formato HDF5 apresentou bom desempenho para leitura e escrita e é adequado ao uso em contexto do processamento paralelo do LAID. Mas ainda assim, foi demonstrado que o código deve implementar formas de evitar a sobrecargas para evitar degradação do desempenho.

Quanto à paralelização dos algoritmos LAID, foi desenvolvido novo código ou adaptado o dos trabalhos anteriores. Posteriormente foi testado no INCD, seguindo um protocolo de teste para produzir os resultados apresentados. Merece destaque a redução de 147 minutos da versão sequencial para 26 minutos da versão paralela de um *dataset* com 1000k de atributos, o que representa um tempo aceitável e uma redução de aproximadamente 82% do tempo anterior.

No entanto, em relação a este tópico de paralelização, nem todos os objetivos foram alcançados. A paralelização passo a passo não ficou concluída, ficou em falta a etapa 3, mas mesmo neste caso, foi possível descrever o que se acredita ser a solução que certamente passará pelo recurso ao uso do modo colaborativo do MPI.

Agradecimentos
Este trabalho foi produzido com o apoio da Infraestrutura Nacional de Computação Distribuída - INCD financiada pela FCT e FEDER no âmbito do projeto 01/SAICT/2016 nº 022153.

REFERÊNCIAS

Agrawal, Ankit, and Alok Choudhary. 2016. "Perspective: Materials Informatics and Big Data: Realization of the 'Fourth Paradigm' of Science in Materials Science." APL Materials 4(5). http://dx.doi.org/10.1063/1.4946894.

Andrew Collette and contributors. "H5py - HDF5 for Python." https://docs.h5py.org/en/stable/index.html (Acedido em 2021-05-12).

Apolónia, João, and Luís Cavique. 2019a. "Seleção de Atributos de Dados Inconsistentes Em Ambiente HDF5 + Python Na Cloud INCD." Revista de Ciências da Computação (14): 85–112.

Apolónia, João, and Luís Cavique. 2019b. "Seleção de Atributos Utilizando a Análise Lógica de Dados Inconsistentes (LAID)." Repositório institucional da Universidade Aberta (UAb) (Ciências e Tecnologia / Sciences and Technology). http://hdl.handle.net/10400.2/8122 (Acedido em 2021-05-06).

Azevedo, Ana, and M F Santos. 2008. "KDD, SEMMA AND CRISP-DM : A PARALLEL OVERVIEW." IADIS European Conference Data Mining: 182–185. http://recipp.ipp.pt/handle/10400.22/136%0Ahttp://recipp.ipp.pt/bitstream/10400.22/136/3/KDD-CRISP-SEMMA.pdf.

Blelloch, Guy E., Harsha Vardhan Simhadri, and Kanat Tangwongsan. 2012. "Parallel and I/O Efficient Set Covering Algorithms." Annual ACM Symposium on Parallelism in Algorithms and Architectures: 82–90.

Boros, Endre, Peter L. Hammer, Toshihide Ibaraki, and Alexander Kogan. 1997. "Logical Analysis of Numerical Data." (October).

Cavique, Luís, Armando B. Mendes, Matthias Funk, and Jorge M.A. Santos. 2013. "A Feature Selection Approach in the Study of Azorean Proverbs." In Exploring Innovative and Successful Applications of Soft Computing, Hershey: IGI Global, 38–58.

Cavique, Luís, Armando B. Mendes, Hugo F.M.C. Martiniano, and Luís Correia. 2018. "A Biobjective Feature Selection Algorithm for Large Omics Datasets." In Expert Systems,.

Cavique, Luís, Armando B Mendes, and Mathias Funk. 2011. "Logical Analysis of

Inconsistent Data (LAID) for a Paremiologic Study." In Processing 15th Portuguese Conference on Artificial Inteligence, EPIA.

Chakravarty, Sreejit, and Ajay Shekhawat. 1992. "Parallel and Serial Heuristics for the Minimum Set Cover Problem." The Journal of Supercomputing 5(4): 331–45.

Ec.europa.eu. 2021. "High Performance Computing." European Commission. https://ec.europa.eu/digital-single-market/en/high-performance-computing.

Han, Jiawei, Micheline Kamber, and Jian Pei. 2012. Data Mining: Concepts and Techniques. Third Edit. Elsevier Inc.

INCD - Infraestrutura Nacional de Computação Distribuída. "INCD User Documentation." https://wiki.incd.pt/books (Acedido em 2021-04-03).

MPI-Forum. "MPI Standard." https://www.mpi-forum.org (Acedido em 2021-06-18).

Pacheco, Peter. 2011. An Introduction to Parallel Programming An Introduction to Parallel Programming. Elsevier Inc.

Pawlak, Zdzisław. 1991. 4 Kluwer Academic Publishers Rough Sets: Theoretical Aspects of Reasoning about Data. Boston: Kluwer Academic Publishers.

Pentland, Alex. 2013. "The Data-Driven Society." Scientific American 309(4): 78–83. http://dx.doi.org/10.1038/scientificamerican1013-78.

Rastogi, Shubhangi, and Hira Zaheer. 2016. "Significance of Parallel Computation over Serial Computation." International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016: 2307–10.

Rissino, Silvia, and Germano Lambert-Torres. 2009. "Rough Set Theory - Fundamental Concepts, Principals, Data Extraction, and Applications." Data Mining and Knowledge Discovery in Real Life Applications (February).

Tony Hey, Stewart Tansley, and Kristin Tolle. 2009. The Fourth Paradigm. 2009th ed. eds. Tony Hey, Stewart Tansley, and Kristin Tolle. Microsoft Research. http://fourthparadigm.org.

Talia, Domenico. 2019. "A View of Programming Scalable Data Analysis: From Clouds to Exascale." Journal of Cloud Computing 8(1).

The HDF Group. "HDF5 High Performance Data Software Library and File Format." https://www.hdfgroup.org/solutions/hdf5 (Acedido em 2021-05-13).

Witten, Ian H., Eibe Frank, Mark A. Hall, and Christopher J. Pal. 2017. Data Mining: Practical Machine Learning Tools and Techniques. Fourth Edi. Elsevier Inc.

Zhang, Jun et al. 2020. “Data-Driven Computational Social Science: A Survey.” Big Data Research 21: 100145. https://doi.org/10.1016/j.bdr.2020.100145.

Zheng, Weimin. 2020. “Research Trend of Large-Scale Supercomputers and Applications from the TOP500 and Gordon Bell Prize.” Science China Information Sciences 63(7): 1–14.

Paulo Morgado, Programador desde 1991, participou em dezenas de projetos de desenvolvimento de software de gestão. Licenciado em Geografia, variante de Sistemas de Informação Geográfica em 2015 pelo IGOT-UL. Obteve o grau Mestre em Informação e Sistemas Empresariais pelo IST-UL e UAb em 2021. Tem como áreas de interesse, os SIG, Computação na CLOUD e “Machine learning”.

Luís Cavique, Professor Auxiliar no Departamento de Ciências e Tecnologia (DCeT), Secção de Informática, Física e Tecnologia (SIFT). Licenciado em Engenharia Informática em 1988 pela FCT-UNL. Obteve o grau Mestre em Investigação Operacional e Eng. Sistemas pelo IST-UTL em 1994. Obteve o grau de Doutor em Eng. Sistemas pelo IST-UTL em 2002. Tem como áreas de interesse, a intersecção da Informática com a Engenharia de Sistemas designadamente a área de “Data Mining".