Computação de Dados em Larga Escala
Projeto Final - Grupo 10

Aluno: Ana Oliveira
Curso: *MEIM*
ISEL, Portugal
Nº Aluno: 39275
Email: A39275@alunos.isel.pt

Aluno: Eduardo Santos
Curso: *MEIM*
ISEL, Portugal
Nº Aluno: 40610
Email: A40610@alunos.isel.pt

***Resumo*—A análise de dados biométricos é cada vez mais comum, nos dias de hoje. Através de dispositivos móveis, como *wearables*, telemóveis e computadores portáteis, ou equipamentos médicos são recolhidos dados de forma contínua, como o electrocardiograma (ECG). A recolha destes dados produz uma vasta quantidade de dados que carece de análise para os mais diversos contextos, como deteção de doenças cardíacas ou acompanhamento do rendimento desportivo, por exemplo. Este projeto insere-se no contexto da análise massiva e inicial de dados de sinais de electrocardiogramas. Analisamos processos como a filtragem do sinal, segmentação em *templates* e deteção de outliers, explorando a framework Hadoop e o modelo de programação MapReduce, para lidar com grandes volumes de dados de forma distribuída e eficiente, através das bibliotecas *Python MRJob* e *biosppy*.**

Index Terms*—*ECG*, *Big Data*, *Hadoop*, *MapReduce*, *Mrjob*, *biosppy

I. Introdução

Nos últimos anos, o avanço tecnológico tem permitido a recolha de uma vasta quantidade de dados biométricos através de dispositivos *wearables*, telemóveis e equipamentos médicos. Estes dispositivos são capazes de adquirir dados fisiológicos, como o ritmo cardíaco, a temperatura corporal e os níveis de atividade, produzindo uma enorme quantidade de dados em tempo real. O eletrocardiograma (ECG) é o sinal mais utilizado para a análise de patologias cardíacas, sendo aplicado em diagnósticos médicos, biometria e investigação.

A aquisição de um sinal de ECG, através de um *wearable*, durante 24 horas de um indivíduo, com uma frequência de 1000 Hz, assumindo uma codificação de 8 bits e desconsiderando parâmetros adicionais, ocupa cerca de 1 Gbyte de dados. Considerando que, num cenário prático, a aquisição do sinal é realizada para mais que um indivíduo, a quantidade de dados adquiridos aumenta consideravelmente. Assim, o processamento destes dados é considerado um problema de *Big Data* [1].

Este projeto ambiciona processar o tratamento inicial para dados de sinais ECG, utilizando técnicas de pré-processamento como a filtragem do sinal, segmentação do sinal em *templates* e detecção de *outliers*. Recorreremos à *framework Hadoop* e às bibliotecas *Python MRJob* e *biosppy* para o desenvolvimento de um sistema distribuído, escalável e capaz de lidar com grandes volumes de dados de ECG.

A. ECG

O eletrocardiograma (ECG) é o processo de registo da atividade elétrica do coração, durante um período de tempo usando eletrodos colocados no corpo de um paciente. Os eletrodos detectam pequenas mudanças elétricas na pele resultantes da despolarização do músculo do coração durante cada batimento cardíaco.

Através de electrodos, colocados em locais adequados no paciente, o sinal ECG é recolhido. A magnitude global do potencial elétrico do coração pode ser medida a partir de diferentes angulos, *leads* e é registada ao longo de um determinado período de tempo.

O complexo QRS é o termo atribuído à combinação de três gráficos de deflexões vistos num eletrocardiograma normal (ECG). É normalmente a parte central e a mais óbvia visualização do traço e representa a despolarização dos ventrículos direito e esquerdo do coração humano. As ondas Q, R e S ocorrem sequencialmente podendo não ser captadas em todas as *leads*. No entanto, uma vez que refletem um evento único são geralmente consideradas em conjunto. A onda Q corresponde a todo o desvio para baixo após uma onda P. Uma onda R segue como uma deflexão para cima, e uma onda S corresponde a qualquer desvio para baixo após uma onda R. A onda T segue a onda S, e em alguns casos, uma onda adicional U segue uma onda T. A figura 1 apresenta o complexo QRS.

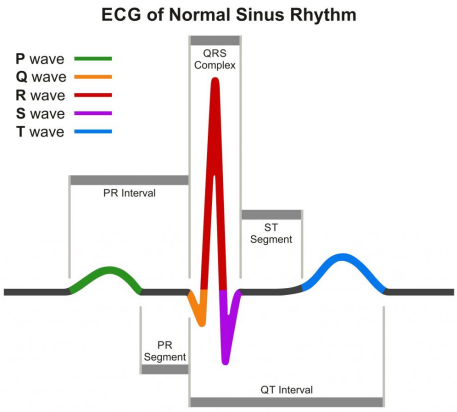

Figura 1. Complexo QRS

O dataset utilizado no projeto foi produzido pela empresa CardioID [2] que através de um dispositivo realizou a aquisição de sinais ECG de diversos indivíduos. A identificação de cada indivíduo encontra-se anonimizada no dataset fornecido. O dataset contém informações como a data de aquisição, a frequência de amostragem do sinal e a referência anonimizada do indivíduo, vem como o sinal ECG exclusivamente para a Lead 1.

B. Haadoop

O *Hadoop* [3] é uma *framework* de código aberto que permite o processamento e armazenamento de grandes quantidades de dados de forma distribuída, em que os dados são divididos e processados simultaneamente em vários computadores. O Hadoop é composto por três componentes principais:

1) **HDFS - Hadoop Distributed File System:**
É um sistema de ficheiros distribuído, concebido para ser executado em *hardware* comum e de baixo custo. É um sistema altamente tolerante a falhas e foi criado para ser utilizado em aplicações que lidam com grandes volumes de dados [4].

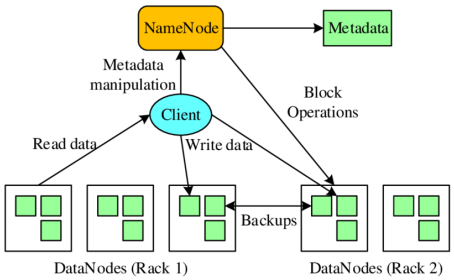

Figura 2. Arquitectura HDFS

Tem uma arquitetura *master/slave* (Figura 2), composta por um único *NameNode* responsável pela gestão do espaço de nomes do sistema de ficheiros e pelo controlo de acesso dos clientes. O *NameNode* executa operações de *namespace*, como abrir, fechar e renomear ficheiros e diretórios, além de determinar o mapeamento de blocos para os *DataNodes*. Por sua vez, os *DataNodes*, gerem os pedidos de leitura e escrita dos clientes e realizam a criação, a remoção ou a réplica de blocos conforme as instruções do *NameNode*.

2) **MapReduce:**
O *MapReduce* é um modelo de programação distribuída para processar grandes volumes de dados de forma paralela num *cluster* de computadores. Divide a tarefa, *job*, em duas fases principais: *Map* e *Reduce*.
Na fase *Map*, os dados de entrada são divididos em pares chave-valor, processados pelos *mappers* de forma a gerar um conjunto intermédio de pares chave-valor.
Na fase *Reduce*, estes pares intermédios são agrupados através da chave. Os *reducers* agregam os valores associados a cada chave de forma a produzir uma saída final. Cada *reducer* produz a sua própria saída.

A figura 3 representa o modelo geral do modelo *MapReduce*.

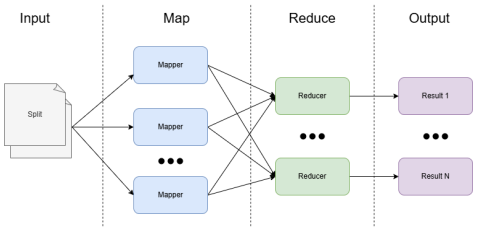

Figura 3. Arquitectura MapReduce

3) **YARN (Yet Another Resource Negotiator):**
O *YARN* (*Yet Another Resource Negotiator*) é uma componente essencial do *Hadoop* que gerencia os recursos num *cluster* e coordena a execução de tarefas, permitindo que múltiplas aplicações, como *MapReduce*, *Spark* e outras *frameworks* de *big data*, sejam executadas no mesmo *cluster*.
É composto por dois componentes principais:
   - **ResourceManager:** gerencia os recursos globais do *cluster* e decide de que forma os deve alocar a diferentes aplicações.
   - **NodeManager:** monitoriza a utilização dos recursos em cada *node* do *cluster*, reportando ao *ResourceManager*.

   Com *YARN*, o *Hadoop* se torna mais flexível e escalável, suportando múltiplos tipos de carga de trabalho.

C. MRJob e BioSppy

O *MRJob* é uma biblioteca, em Python, que facilita o desenvolvimento de *jobs MapReduce* para serem executados em diversos ambientes, como *Hadoop* ou *Amazon EMR*, pois abstrai a necessidade de interagir diretamente com a API Java do Hadoop.

A escolha do *MRJob* recaiu na integração com a biblioteca de processamento de sinais de electrocardiogramas *BioSPPy*, bem como a possibilidade de permitir utilizar o *Hadoop* tirando partido da familiaridade com a linguagem de programação *Python*. Embora o *MRJob* permita simplificar o desenvolvimento de *jobs MapReduce* contém algumas limitações, como a definição de *inputFormat* e *outpuFormat* e cache distribuída como iremos detalhar no capítulo de implementação.

A biblioteca *BioSPPy* é uma *toolbox* para processamento de bio-sinais em *Python*. Oferece um conjunto de ferramentas para a análise de sinais, incluindo filtragem, deteção de *outliers* e técnicas de *clustering*.

II. Trabalho Relacionado

Nas últimas décadas, diferentes de abordagens têm sido exploradas no que concerne o processamento de grandes volumes de dados biométricos, em particular, com os dados de ECG. O artigo (Kerk Chin Wee, Mohd Soperi Mohd Zahid, 2015) [5] propõe uma solução baseada em computação

na núvem e *MapReduce* para a otimização na análise de grandes volumes de dados de ECG. O recurso da *framework MapReduce* permite distribuir o processamento entre múltiplas máquinas virtuais num *cluster*. Esta abordagem está alinhada com os objetivos do trabalho proposto. No entanto, procuramos obter uma solução compatível com *Python* e escalável para diversos tipos de contextos. Enquanto a proposta do artigo [5] concentra-se numa abordagem baseada em *MapReduce* para ambientes de computação em nuvem, este projeto procura a integração diferentes tecnologias.

III. Implementação

Neste capítulo detalharemos a implementação do projeto proposto para o processamento de dados de ECG, utilizando *MapReduce*.

O sistema foi desenvolvido com recurso à biblioteca *MrJob* e é composto por três componentes principais:

- Leitura dos dados (sinais ECG);
- Segmentação dos sinais em templates;
- Deteção de *outliers*.

Adicionalmente, o sistema realiza a filtragem dos dados e realiza a contagem dos templates produzidos. A arquitetura geral do sistema é apresentada na Figura 4.

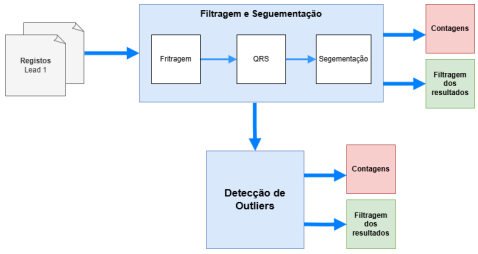

Figura 4. Arquitetura Geral

A. Tarefas (Jobs)

O projeto é composto por diversos *jobs* responsáveis pelo tratamento de cada um dos blocos representados na Figura 4. Cada *job* tem um *script* em *bash* que permite que possa ser executado independentemente dos restantes *jobs*. Para além disto, foi desenvolvido um *script* responsável pela execução sequencial de todos os *jobs* demonstrando o fluxo do sistema. A lista de *jobs* disponíveis encontra-se representada na Figura 5.

```
==============================
         MAIN MENU
==============================
1. Template Segmentation
2. Template Counter
3. Filter Templates
4. Outlier Detection
5. Outliers Counter
6. Filter Outliers
0. Exit
==============================
Enter your choice (0-6):
```

Figura 5. Main page

Adicionalmente, cada *Job* tem associado um ficheiro de configuração, no qual são definidas as configurações do *Hadoop*, como o número de *reducers*, *codecs* e o tamanho mínimo dos blocos para SPLIT dos ficheiros. Além disto, os *jobs* partilham diversos vários ficheiros distribuídos em *cache*. Existe ainda um ficheiro de parâmetros de configuração em JSON, que contém atributos relevantes para a execução dos *jobs*, consoante as suas funções.

B. Leitura dos dados

Os dados de *input* do sistema apresentam-se em formato de texto, onde cada ficheiro contém a aquisição de um sinal de ECG ao longo de um período de tempo indeterminado. A referência anónima do utilizador encontra-se no nome do ficheiro, e dentro dele, um cabeçalho contém informações gerais da aquisição, como a data de início da aquisição, a frequência de amostragem e as *labels* associadas. Essas *labels* consistem no *timeStamp*, o valor do sinal no momento específico da aquisição, e o Lead on Detection (LoD).

Numa primeira fase, os ficheiros de *input* não podem ser partidos devido à existência do cabeçalho do ficheiro. Assim, estes ficheiros são configurados como *non-splittable*. Esses ficheiros são, então, tratados pelo *Hadoop* organizando-os em blocos de 4 MB. Para conseguir tratar as aquisições, o *Hadoop* atribui um ficheiro por *mapper*. O formato de saída definido permite que os ficheiros produzidos por cada *mapper* possam ser partidos pelos *jobs* seguintes. Por defeito, o *MRJob* trata todos os ficheiros como sendo *splittable*. No entanto, é possível definir um valor superior ao tamanho do ficheiro evitando o *split* do ficheiro.

C. Template Segmentation

A segmentação dos *templates* é definida por três blocos principais, filtragem do sinal, deteção de picos e segmentação dos *templates*. Como suporte a estas operações, utilizámos a biblioteca *BioSPPy*, que fornece as ferramentas necessárias para o processamento de cada um destes blocos.

A filtragem do sinal ECG é fundamental para a remoção do ruído proviniente da aquisição do sinal, frequentemente provocada pela corrente elétrica existente no local da aquisição dos dados, introduzindo interferências de alta frequência.

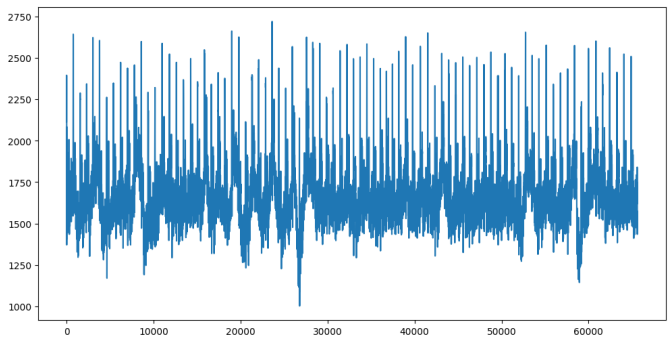

Figura 6. Sinal Ecg Raw

A identificação do complexo QRS é conseguida através da deteção dos picos R, que permitem identificar os batimentos cardíacos. Através destes picos é possível segmentar o sinal em ciclos individuais que facilitam o processo de análise e extração de características. Este processo trata-se de um processo fiducial existindo outras possíveis abordagens na obtenção de características relevantes.

Para a concretização deste processo, foi atribuído um *Job* responsável pela leitura dos ficheiros, filtragem o sinal, realização da deteção de picos e segmentação do sinal, produzindo um ficheiro de *output* composto por um par chave-valor. A chave é composta pelos dados do utilizador, a frequência e o índice de ordem, e o valor corresponde ao respetivo *template* segmentado.

D. Outlier Detection

Os *outliers* consistem em inconsistências, valores atípicos ou observações que "fogem"ao padrão pretendido. Dependendo do contexto, estes valores podem ter diferentes significados. Para o caso dos sinais ECG estes valores podem ser relevantes, podendo, por exemplo, representar patologias.

Mais uma vez, com recurso à biblioteca *BioSPPY* foi utilizado o algoritmo *DMean* para a identificação dos *outliers*. Este algoritmo agrupa os dados em *clusters* e com recurso à distância euclideana à média de cada *cluster* determina os *outliers*. O limiar de decisão considerado é determinado pelos parâmetros *alpha* e *beta*.

Também este processo é conseguido através de um *job* dedicado que é responsável pela deteção dos *outliers*. Este *job* produz uma nova chave, composta pela designação do utilizador, data, frequência e o tipo (*outlier* ou não), e o respetivo valor, composto por um array de templates.

E. Filtragem e counters

Em determinados contextos poderá ser útil consultar o número de *templates* produzidos para um determinado utilizador, numa determinada data, de um tipo e com uma frequência específica. Para isso, foram definidos vários *jobs* responsáveis pelas contagens em cada um dos blocos, aproveitando as chaves compostas atribuídas nos blocos principais.

IV. Resultados

Neste capítulo apresentamos os resultados obtidos no desenvolvimento do sistema proposto, focando a análise da segmentação de *templates* e identificação de *outliers*.

A. Counters

A contagem dos *templates* foi conseguida através de dois *jobs*, um para a contagem de *templates* com base nos resultados da segmentação e outro com base na deteção de *outliers* (Tabelas I e II). Esta abordagem permitiu uma análise estruturada dos dados processados para cada etapa do sistema. Observa-se que o uso de *jobs* separados para essas operações para além de simplificar a organização, também facilita na identificação de possíveis inconsistências nos resultados.

Tabela I
EXEMPLOS DE ALGUNS RESULTADOS DA CONTAGEM DE TEMPLATES

Utilizador	Nº de Templates
u01	167
u02	160
u03	183
u04	279
u05	181
u06	231
u07	154

Tabela II
EXEMPLOS DE ALGUNS RESULTADOS DA CONTAGEM DE OUTLIERS

Utilizador	Nº de Templates Válidos	Nº de Outliers
u01	137	30
u02	125	35
u03	149	34
u04	225	54
u05	154	27
u06	170	61
u07	124	30

B. Templates

Através da filtragem e segmentação dos *templates* é possível analisar, na Figura 7, a sobreposição de todos os *templates* segmentados. Verifica-se que o sinal ECG tende a manter os mesmos padrões ao longo do tempo, embora possam ocorrer variações devido a fatores como stress, condições de saúde ou movimentos realizados durante a aquisição, entre outros aspetos.

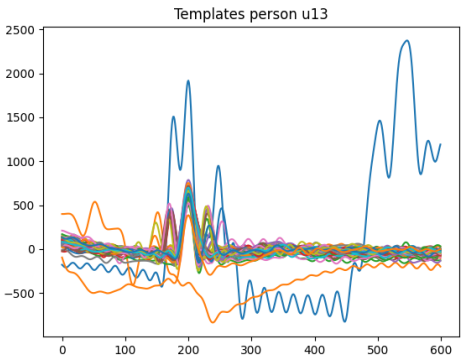

Figura 7. Exemplo Filtragem Template Segmentado

C. Outliers

A deteção dos *outliers* detetados permite remover os mesmos mantendo apenas os *templates* essenciais a análises futuras. Podemos observar através da comparação da Figura 7 e da Figura 8 uma clara melhoria no que concerne os *templates* que melhor caracterizam o sinal do indívido 13, tendo sido removidos os *outliers* identificados. A Figura 9 apresenta os *outliers* detetados na análise dos sinais do indivíduo 13.

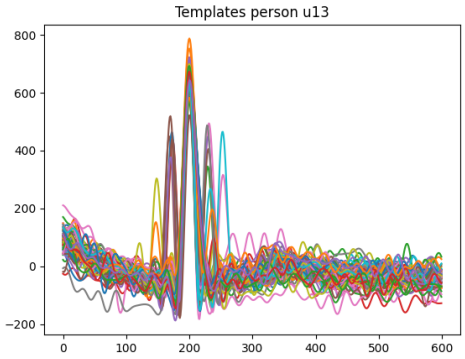

Figura 8. Exemplo Filtragem Outliers Válidos

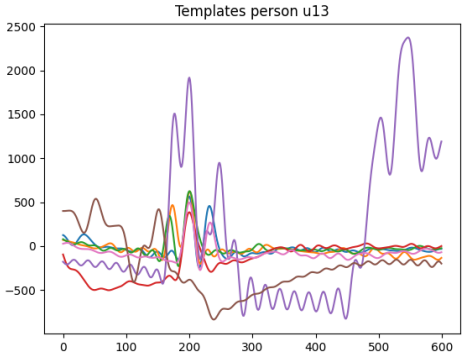

Figura 9. Exemplo Filtragem Outliers

V. Conclusões e Trabalho Futuro

O projeto desenvolvido resultou num sistema de processamento de sinais de electrocardiograma (ECG) no contexto de *Big Data*. A utilização do *framework MapReduce* em conjunto com a biblioteca *MrJob* permitiu implementar uma solução distribuída capaz de realizar tarefas como filtragem de sinais, segmentação de *templates* e deteção de *outliers*, de forma estruturada e paralelizada.

No entanto, algumas limitações foram observadas, nomeadamente a impossibilidade de particionar ficheiros devido à estrutura dos cabeçalhos, aliada à customização do *inputFormat* e *outputFormat* através do protocolo binário no *MrJob*, apresentou várias restrições no que diz respeito ao *split* dos ficheiros. Apesar da solução proposta ter contornado essas limitações, esse aspeto pode ser significativamente melhorado com a adoção de outros formatos de ficheiros, como os utilizados em implementações em Java.

Para trabalhos futuros, sugerimos a implementação de técnicas de *clustering* para agrupar *templates* semelhantes, além da exploração de novos algoritmos de deteção de *outliers* mais robustos. Recomendamos a aplicação de novas funcionalidades, aproveitando a elevada escalabilidade do sistema, para expandir a sua capacidade de lidar com diferentes tipos de dados e cenários, tornando-o mais flexível e adaptável a diversos contextos.

Referências

[1] Big Data. https://www.sciencedirect.com/topics/computer-science/big-data-problem.
[2] CardioID. https://www.cardio-id.com/.
[3] Hadoop. https://pt.wikipedia.org/wiki/Hadoop.
[4] HDFS. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[5] Kerk Chin Wee and Mohd Soperi Mohd Zahid. Cloud computing for ecg analysis using mapreduce. In *2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, pages 115–120, 2015.