Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Estatística
Programa de Especialização em Data Science e Big Data

Kalleby Lee Araujo Ramos

Machine Learning para processos em Banco de Dados

Curitiba
2020

Kalleby Lee Araujo Ramos

Machine Learning para processos em Banco de Dados

Monografia apresentada ao Programa de Especialização em Data Science e Big Data da Universidade Federal do Paraná como requisito parcial para a obtenção do grau de especialista.

Orientador: Eduardo Cunha de Almeida

Curitiba
2020

Especialização em Data Science e Big Data
Universidade Federal do Paraná
dsbd.leg.ufpr.br

Machine Learning para processos em Banco de Dados

Kalleby Lee Araujo Ramos[1]
Simone Dominico[2]
Eduardo Cunha de Almeida[3]

Resumo

Extract Transform Load (ETL) é a sistematização do tratamento e limpeza dos dados gerados através dos diversos sistemas organizacionais para a sua posterior inserção, geralmente em *Data Warehouse* ou *Data Mart*. A urgência para processos de ETL mais ágeis é eminente. Contudo, em negócios com uma estrutura já arquitetada, a migração para arquiteturas recentes e escaláveis (*Cloud*) pode tornar a implantação um trabalho difícil. Neste contexto, o objetivo é melhorar a agilidade nos processos de ETL utilizando *Machine Learning* (Aprendizado de máquina). Para isto, foi feito um estudo do comportamento do banco de dados para detectar padrões de uso de processamento e taxa de transferência da memória e agendar vários processos de ETL. A agilidade dos processos ETL será maior devido à quantidade de vezes que o processo será realizado durante o dia. Normalmente os processos de ETL são realizados apenas uma vez por dia devido à concorrência de processamento, ao detectar padrões de comportamento os processos ETL poderão ser agendados mais de uma vez ao dia. Dessa maneira, a latência de atualização dos dados será otimizada e problemas como atraso de informações, atualmente em D-1 (atraso de 1 dia) poderão ser evitados.
Palavras-chave: ETL, data warehouse, machine learning

Abstract

Extract Transform Load (ETL) is the systematization of the treatment of data generated by many organizational systems, usually stored in a Data Warehouse or Data Mart. The urgency for a more resource efficient ETL process is eminent. In this context, the objective is to improve the efficiency of the ETL processing using an adapted Machine Learning clustering algorithm. For this, we study the behavior of the ETL resource usage from many production ETL processes. Normally the ETL process is executed once a day due to resource contention issues. With our ML algorithm in a production scenario, the ETL processes can be scheduled to execute more than once a day. Thus, the latency of data update to feed analytic systems can be optimized and problems like information delay, currently in D – 1 (delay of one day) can be avoided.
***Keywrods**: ELT, datawarehouse, machine learning*

1 Introdução

Atualmente, com o avanço da tecnologia o volume de dados cresce exponencialmente em diferentes setores como manufatura, serviços financeiros, educação, comércio, entre outros. Os dados são provenientes de diferentes fontes e impulsionam a utilização de sistemas como *data warehouse*. Um *data warehouse* é uma base de dados especializada que incorpora dados oriundos de diversos bancos de dados operacionais. Os dados de um *data warehouse* são utilizados para tomada de decisões estratégicas e precisam ser atualizados para garantir a precisão das decisões de uma organização. O processo de alimentação de um *data warehouse* é um processo complexo que envolve diferentes etapas como a extração, transformação e carga dos dados.

As etapas do processo de alimentação do *data warehouse* são conhecidas como processos de ETL (*Extract, Transformation and Load - ETL*) [1]. Segundo [2], os processos de ETL demandam 70% do tempo e recurso para extrair os dados de uma fonte específica e armazenar em um *data warehouse*. Para a tomada de decisões com informações de um *data warehouse* em tempo real, os processos de ETL podem ser feitos enquanto o banco de dados está em uso. Além disso, o *data warehouse* é utilizado para a geração e análise de relatórios. Para um efetivo gerenciamento de um *data warehouse* é necessário gerenciar e agendar tarefas de processamento e manutenção. Entretanto, os processos ETL em um *data warehouse* podem causar um impacto significativo no desempenho, na qualidade/integridade dos dados e na divisão de recursos computacionais.

Outro ponto importante para o desempenho dos processos ETL é o crescimento no volume de dados da organização. À medida que o volume de dados cresce os processos de ETL devem ser redimensionados para apresentar um desempenho aceitável evitando gargalos de latência no acesso aos dados. No entanto, muitas organizações de pequeno e médio porte não conseguem adequar sua infraestrutura para armazenar e extrair in-

[1]Aluno do programa de Especialização em Data Science & Big Data, kallebyramos@hotmail.com.
[2]Aluno de Doutorado do Programa de Pós-Graduação em Informática, sdominico@inf.ufpr.br.
[3]Professor do Departamento de Informática - DINF/UFPR.

formações dos dados em tempo real com baixa latência de dados. Além disso, gerenciar as tarefas em um *data warehouse* garantindo dados atualizados depende da frequência de manutenção e os processos de ETL precisam ser realizados sem afetar atividades de análise e geração de relatórios.

Neste sentido, esse artigo busca responder a seguinte pergunta: Como melhorar a latência (tempo de atualização das informações) dos dados usando *Machine Learning* para agendamento de processos de ETL?

Todas as tarefas realizadas em uma máquina incluem principalmente o uso de *hardware*. Para iniciar uma tarefa é necessário ter processamento, memória RAM e em alguns casos específicos disco rígido. Em um processo de ETL não é diferente, são utilizados recursos computacionais para realizar cada etapa do processo de ETL.

Neste artigo, apresentamos um modelo de *Machine Learning* (ML) de clusterização (*K-Means*). Através da clusterização pretendemos classificar em *clusters* o uso de recursos computacionais e identificar janelas para o agendamento dos processos de ETL. Os *clusters* representam o estado do banco de dados como descrito em [3]. O objetivo principal é otimizar o processamento dos dados e reduzir a latência dos dados durante os processos de ETL.

O artigo está organizado da seguinte maneira. A Seção 2 apresenta a motivação e os conceitos importantes para o desenvolvimento deste artigo. A Seção 3 discute trabalhos relacionados. A Seção 4 detalha o modelo proposto. A Seção 5 apresenta um estudo comparativo através de experimentos, enquanto a Seção 6 conclui o trabalho e apresenta os trabalhos futuros.

2 Motivação e conceitos fundamentais

Nesta seção apresentamos a motivação e alguns conceitos fundamentais para o desenvolvimento deste trabalho.

2.1 Motivação

Normalmente o gerenciamento de um *data warehouse* envolve a coordenação de diferentes atividades e sistemas conectados. Automatizar os processos de ETL pode otimizar o uso dos recursos disponíveis no servidor e evitar gargalos de recursos computacionais. Neste sentido, um algoritmo de ML pode detectar janelas de agendamento dos processos de ETL.

Um algoritmo de análise do uso de processamento e de memória do banco de dados, pode obter informações do uso dos recursos computacionais do servidor em que o banco de dados foi alocado. Com essas informações é possível treinar um modelo de ML para criar *clusters*, separando-os por uso de recursos computacionais, como por exemplo: baixo uso, médio uso e alto uso. Para tal feito, o modelo *K-means* de ML se enquadra, pois basicamente sua lógica permite separar os dados em clusters definidos a partir de um centroide inicial.

Um modelo de ML basicamente é uma composição de fatores até chegar em um resultado final, chamado de modelo ajustado. Ou seja, sempre que é especificado um modelo de treinamento de máquina, pode-se interpretar como uma composição de fatores como algoritmos, regras, testes, treinamentos, entre outros. Já um algoritmo, como citado anteriormente, é uma etapa até chegar no modelo ajustado.

Ao adaptar esse modelo de aprendizado de máquina, podemos obter a informação necessária para futuramente analisar e agendar rotinas automáticas baseadas nos resultados obtidos pelo algoritmo que foi utilizado para o treinamento do modelo, minimizando assim o problema de processamento de dados e inúmeros outros problemas decorrentes de uso do banco de dados. Diante disso, o ML pode ajudar a detectar padrões de uso de recursos computacionais e fornecer informações para o agendamento dos processos de ETL.

Para a análise do uso de recursos computacionais, a taxa de transferência de memória e uso de processamento podem exibir uma tendência de utilização sendo possível planejar e agendar um ou mais processos de ETL no banco de dados. Neste contexto , o uso desses recursos computacionais como parâmetro de treinamento do modelo de ML pode reconhecer essas tendências de utilização e fornecer informações para o agendamento.

2.2 *Clusterização de dados*

A clusterização de dados é o processo de particionar ou detectar "outliers"para encontrar um padrão, ponto ou objeto [4]. O particionamento pode ser feito com inúmeros métodos de clusterização, alguns deles são baseados em centroide, densidade, distribuição ou até mesmo por hierarquia.

Os métodos de clusterização, são divididos em dois grupos, similaridade e distância, esses dois grupos são basicamente a forma com que o algoritmo faz seu cálculo de clusterização. Distância (dissimilaridade) e similaridade são a base para a construção de um algoritmo de clusterização. Para os dados de recursos quantitativos, a distância é mais adequado para reconhecer a relação entre os dados. Enquanto, a similaridade é preferível quando são utilizados recursos de dados qualitativos [5].

2.3 K-Means

Entre os algoritmos de clusterização o K-means é um algoritmo que classifica informações semelhantes de acordo com os próprios dados [6]. O K-means é um algoritmo não supervisionado em ML, esses tipos de algoritmos não precisam de nenhum rótulo, separação ou pré classificação para realizar a classificação. O algoritmo faz o trabalho de inferir a clusterização utilizando apenas o cálculo da distância de cada dado do seu centroide.

Um centroide é como um protótipo para um *cluster*. Na maioria das técnicas, pontos aleatórios do conjunto

de dados são escolhidos para serem os centroides iniciais. Na sequência, cada instância desses dados é atribuída ao centroide mais próximo. Nas demais iterações a posição dos centroides é calculada através da distância média entre todos os pontos atribuídos aquele centroide na última iteração [7]. O algoritmo termina o cálculo quando a posição dos centroides não é mais modificada ou a distância da mudança é menor que um valor predefinido.

Os algoritmos de clusterização precisam de uma métrica, especificamente da distância entre dois dados. O K-means é fortemente tipado, ou seja, se um dado foi designado para um *cluster*, ele será exclusivamente daquele *cluster*, não existe a possibilidade de um dado estar em dois *clusters* ao mesmo tempo. O algoritmo aprende a posicionar os dados em seus respectivos *clusters* utilizando a soma total da distância euclidiana entre o respectivo dado e o seu centroide, levando em consideração sempre a menor distância. Esse cálculo é baseado na seguinte fórmula matemática:

$$min_{C_1,...,C_k,\mu_1,...,\mu_k} \sum_{n=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2 \quad (1)$$

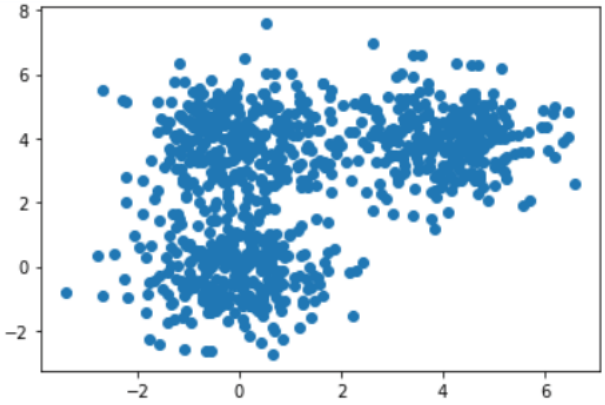
Cada um dos *clusters* C contém uma sub massa de dados, que nada mais é do que um conjunto de massa dentro de outro conjunto de massa. Por exemplo, eu tenho o conjunto X de massa de dados e dentro desse conjunto X eu encontro o X_a, X_b ,X_c e assim por diante. O centroide do *cluster* i é igual a média da soma de todos os dados que estão no *cluster* e pode ser representado pela seguinte fórmula:

$$\mu_i = \sum_{x \in C_i} x/n_i \quad (2)$$

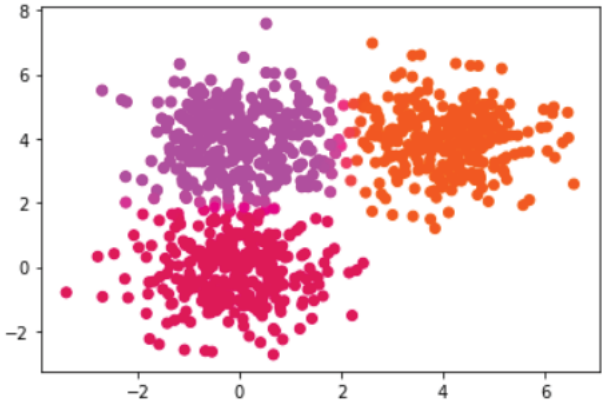
Onde n_i mostra o número de dados que estão no *cluster* i.

A Figura 1 (a) mostra como o k-means trabalha em uma massa de dados gerada aleatoriamente pela distribuição Gaussiana. Os resultados, são sensíveis e podem mudar de acordo com o número de *clusters* desejados, essa parâmetro é passado dentro do algoritmo. A Figura 1 (b) mostra como o K-means retorna o resultado dos *clusters* previamente definidos. E possível notar a clara diferenciação dos dados.

Também conhecido como distribuição normal é uma curva simétrica em torno do seu ponto médio.



(a) Dados Gerados aleatoriamente



(b) *Clusters* encontrados pelo k-means

Figura 1: Primeira figura mostra dados aleatórios gerados pelo algoritmo em python. Já a segunda figura mostra como fica a massa de dados após o K-means classificar os *clusters*.

3 Trabalhos relacionados

Neste artigo o principal objetivo é detectar padrões de uso de recursos computacionais para realizar o agendamento dinâmico de processos ETL. Nesta Seção discutimos alguns trabalhos relacionados com a automação de processos ETL.

Em [8] os autores apresentam um mecanismo para automatização de processos de ETL em um *data warehouse* usando um *log* que armazena o estado em que se encontra a fonte de dados e um *buffer* que armazena as mudanças no estado da fonte de dados. O *log* é atualizado quando os dados não estão sendo processados, ou seja, o *buffer* não foi atualizado. Em nosso mecanismo, usamos clusterização para identificar através de informações de uso de recursos computacionais para indicar o melhor momento para o agendamento de processos ETL não precisamos de informações diretas do banco de dados.

Em [9] é apresentado um *scheduler* de processos de ETL. Esse *scheduler* analisa relatórios para agendar sua atualização periodicamente. A sequencia de atualização do banco de dados ocorre através de uma consulta que

identifica uma lista de tarefas dentro do *data warehouse*, assim fazendo a execução de atualização de acordo com essa lista. Diferente do nosso trabalho os autores não usam informações de *hardware* e *machine learning* para o agendamento dos processos de ETL.

No artigo descrito em [10] o problema apresentado é semelhante ao que estamos estudando. Entretanto, a solução proposta foi a utilização de duas plataformas de ETL que se integram, o *Informatica* e o *Jenkins*. Combinando as duas plataformas os autores criam um *pipeline* de dados automático e otimizado.

Os processos de *pipeline* de dados ganharam maior relevância na era do *Big Data*. Junto ao grande aumento de fluxo de dados, surgiram novas soluções para tornar eficaz os processos de limpeza e persistência de dados. Porém, existe uma escassez de processos de avaliação e monitoramento dessas novas implementações de *Big Data*. Tendo isso em vista, algumas soluções com ML estão surgindo, como aplicação de séries temporais para criação de classes [11].

4 Machine Leaning para agendamento de processos ETL

Nesta seção apresentamos a metodologia de ML adotada, o banco de dados utilizado para extração dos dados de recursos computacionais e os dados envolvidos no processo de treinamento do modelo de ML.

O método de clusterização foi o escolhido para classificar os dados. Este método permite separar dados em *clusters*, o que é fundamental para a proposta desse artigo. Dentro da clusterização há inúmeras técnicas para execução, a escolhida foi o *K-Means*, ele parte do princípio de definir um dado central e fazer cálculos dos dados ao redor desse centro para definir seus *clusters*.

O banco de dados utilizado para análise dos dados de recurso computacional foi o **MonetDB** (v11.25.5), além da vantagem de ser *open source*, a sua arquitetura favorece grandes bases com um alta requisição de processamento. Além disso, o **MonetDB** foi desenvolvido para cargas de trabalho de *data warehouse* [12].

4.1 Conjunto de dados

Os dados utilizados são de uso de processamento e taxa de tranferência de memória do servidor. Essas informações fornecem uma ampla visão do estado atual de uso de recurso do servidor em que o banco de dados está instalado. Esses dados foram obtidos na execução de um *benchmark* com carga de trabalho analítica nomeado de TPC-H. Os dados foram coletados em um servidor Non-Uniform Memory Access (NUMA) formado por 4 nós com um Quad-Core AMD Opteron 8387 cada, executando a 2,8 GHz. Cada nó Opteron é formado por quatro núcleos com *cache* L1 privado (64 KB) e *cache* L2 (512 KB) e um *cache* L3 compartilhado (6 MB). Este servidor inclui 64 GB de memória principal DDR-2 e disco SATA de 1,8 TB.

4.2 Dados de processamento

Dados de processamento são informações de uma série de atividades executadas ordenadamente dentro do servidor. Quando são realizados qualquer consulta no banco de dados, aumenta o consumo de processamento, pois precisa processar a informação para gerar o resultado. Neste contexto, será usado esse parâmetro para medir se existe a possibilidade de inserir mais tarefas para serem executadas, otimizando assim o uso do processamento de forma cautelosa e efetiva.

4.3 Dados de memória

Os bancos de dados em memória, utilizam a memória RAM para armazenar os dados temporariamente. Quando os dados estão armazenados na memória RAM existe um aumento de performance. O acesso a memória RAM é mais rápido por que a latência de acesso aos dados é menor comparado com o acesso ao disco. Entretanto, um banco de dados em memória quando está processando uma carga de trabalho analítica faz várias requisições para a memória RAM. Neste contexto, usamos a taxa de transferência de memória para analisar o estado do banco de dados e detectar possíveis *clusters*. Combinamos dados de uso de processamento e memória para treinar nosso algoritmo de ML.

5 Análise dos Resultados

Para iniciar os experimentos, os dados foram obtidos através do processamento de máquina e a taxa de transferência de memória dos nós NUMA (NUMA 0, NUMA 1, NUMA 2 e NUMA 3). A matriz aplicada nos testes, mostra para qual *cluster* cada dado da matriz foi alocado, dessa forma, temos uma visão geral da matriz e seus *clusters* por dado, podendo usar como comparação para os dados originais. Os dados de centroide mostrados ajudam a visualização da alocação, a média entre os pontos utilizada na definição do centro e também suas coordenadas no plano.

5.1 Processamento x Segundo

No primeiro experimento apresentado nas Figuras 2,3 e 4, foi executado o *K-means* com o eixo *Y* contando os segundos e o eixo X com os dados de processamento, dividindo tudo em 3 *clusters* de classificação. O resultado não foi satisfatório, porque o *K-means* não conseguiu se adequar a forma com que os dados estavam dispostos. Nesse caso, o eixo com os dados chamados de segundos estavam em uma disposição linear, não se encaixando na proposta que o algoritmo de ML precisa.

5.2 Processamento x Memória

Com o resultado alcançado no primeiro experimento, foi realizado uma análise de métricas que poderiam ser aplicadas para detectar a performance do banco de dados. Como a métrica de tempo é algo linear, não poderia ser

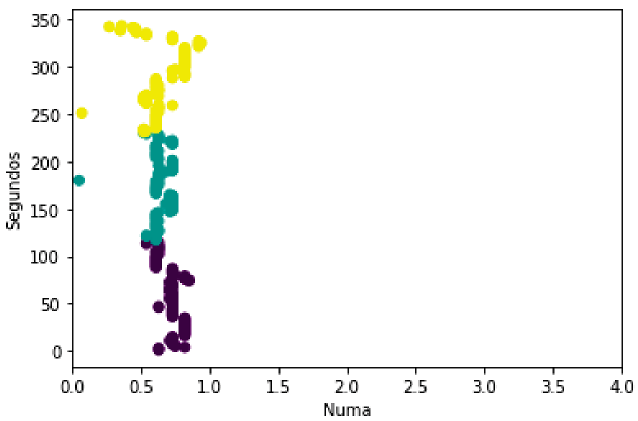


Figura 2: Dados de tempo e processamento no servidor NUMA

```
Matriz
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2]
```

Figura 3: A matriz monstra a qual cluster cada dado foi alocado.

utilizada para o treino do algoritmo de ML. Com isso, a métrica definida foi processamento por taxa de transferência de memória. Além disso, foi alterado as colunas utilizadas para o experimento, ao invés de utilizar a taxa de processamento e taxa de memória separados por nó (NUMA 0, 1, 2 e 3), foi feita a somatória de todos os nós em cada métrica utilizada (total de uso de processamento e taxa de transferência de memória), resultando em duas colunas com os totais. Neste experimento os eixos foram alterados para total de uso de memória e total de uso de processamento (4 nós juntos). Além disso, foi aplicado uma divisão de 3 *clusters* aos dados, os pontos pretos maiores no centro de cada *cluster* são seus respectivos centroides.

Com esse ajuste nos dados podemos observas nas Figuras 5, 6 e 7, temos no eixo Y a taxa de transferência da memória e no eixo X o uso de processamento. Utilizando o *K-means* com classificação de 3 *clusters* nos dados ajustados, o comportamento do algoritmo melhorou, os dados não estão mais de forma linear e a classificação foi efetuada com êxito. Porém, há uma margem de erro na classificação, o número definido de *clusters* não foi satisfatório, pois com apenas 3 *clusters* o intuito de classificação por baixo uso de máquina, médio uso e alto uso não foram satisfeitos. Como pode-se observar na Figura 5 os *clusters* foram corretamente separados, porém, o *insight* que temos observando este resultado é que o *cluster* amarelo (alto uso) não necessariamente representa um alto uso, porque ele tem pontos com processamento e memória com médio uso também. Desta forma, a classificação não foi feita corretamente e foi ne-

```
Localização dos centróides:
[[ 58.5         0.71663793   0.45112069   0.45862069   0.61456897]
 [173.5         0.6422807    0.36236842   0.5972807    0.58157895]
 [287.          0.66584071   0.41415929   0.53628319   0.57451327]]
```

Figura 4: Coordenadas dos centroides de cada *cluster* encontrado com os dados de processamento.

cessário aumentar o número de *clusters* para alcançar o resultado esperado do modelo.

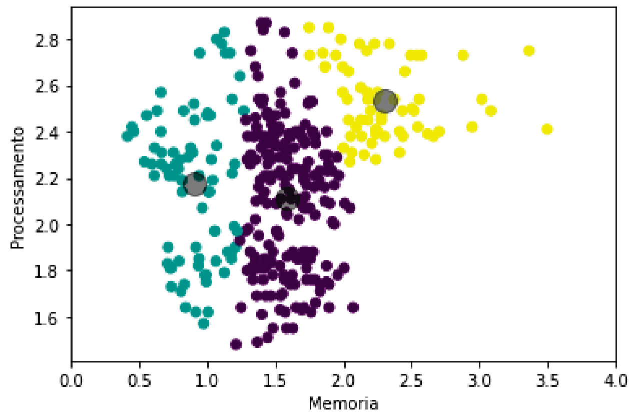


Figura 5: Dados totais dos nós NUMA por processamento e taxa de uso de memória com 3 *clusters*.

```
Matriz
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 1 1 1 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 0 2 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2
 2 2 2 2 2 2 2 2 2 2]
```

Figura 6: A matriz monstra a qual *cluster* cada dado foi alocado.

Após uma segunda rodada de experimentos com inúmeros *clusters* (utilizando progressivamente os clusters de 3 até o número ideal), foi obtido o número ideal de *clusters* para a classificação, totalizando 10 *clusters*. Com essa quantidade de clusters foi possível classificar o uso da máquina de forma mais assertiva, conforme a Figura 8. Para conseguir mudar o número de *clusters* foi necessário apenas passar como parâmetro a quantidade de *clusters* no cálculo do *K-means*, a lógica de aplicação desse parâmetro pode variar de acordo com o código utilizado para o experimento. Neste caso, simplesmente foi passado para uma variável, por exemplo, *cluster* = 10 (número de *clusters*) e posteriormente foi utilizada essa variável como parâmetro na hora de aplicar o algoritmo aos dados.

5.3 Tempo de atualização das informações

Ao analisarmos os resultados dos experimentos, comprovamos que o o resultado do modelo ajudará na redução da latência de informações. Através dos resultados é possível classificar o estado do servidor. Com isso, um

```
Localização dos centróides:
[[2.30593387 2.53548387]
 [0.90180101 2.1756962 ]
 [1.58318911 2.11559406]]
```

Figura 7: Coordenadas dos centroides de cada *cluster* encontrado.

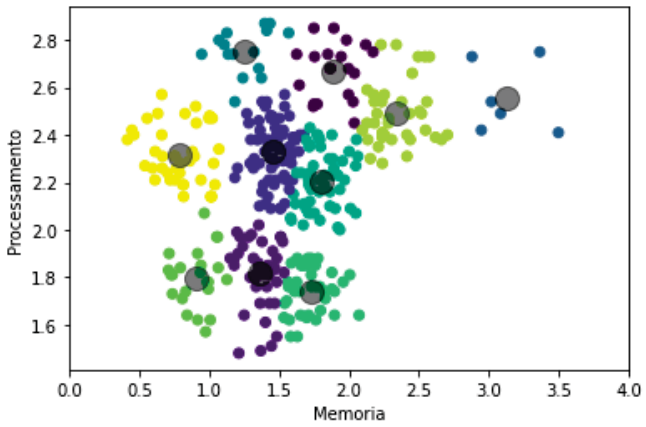


Figura 8: Dados totais dos nós NUMA por processamento e taxa de uso de memória com 10 *clusters*.

processo de ETL que poderia ser desencadeado apenas em horários extraordinários, como fora do expediente de trabalho, pode ser alocado em janelas menores de tempo durante o dia, observando o estado do servidor indicado no modelo.

Com a detecção de uso de recursos computacionais do servidor, utilizando o modelo pode-se definir e agendar o processo de ETL durante o dia, não precisando esperar o fim do expediente. Deste modo, é possível melhorar a latência dos dados de 1 dia para algo em torno de algumas horas dependendo do estado do servidor apontado pelo modelo.

6 Conclusão

Utilizando o *K-means* é possível detectar o comportamento do uso de recursos computacionais no servidor em que o banco de dados está alocado e classificá-lo corretamente. Entretanto, é necessário um maior número de clusters. Com uma pequena quantidade de *clusters* não é possível uma classificação correta.

Com a definição de 3 *clusters* o cenário de pouco processamento e muito uso de RAM acabou ficando como "Alto uso da máquina", conforme a Figura 5 e essa informação não é verídica. Por outro lado, com o uso de 10 *clusters* o cenário de pouco processamento e muito uso de memória ficou em um *cluster* separado e pode ser rotulado como um "Médio uso da máquina" junto a outros *clusters* com o um cenário correspondente como na Figura 8. Sendo assim, obtemos mais de um *cluster* para cada situação, abrangendo melhor e com mais assertividade o estado da máquina em que o banco de dados está alocado.

```
Matriz
[7 7 7 3 3 3 7 7 7 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 1 1 1 1 1 1
 9 9 9 9 9 9 9 9 9 4 4 5 4 4 4 4 4 4 4 4 4 5 8 5 5 8 8 8 8 0 8 8 0 8 0 0 0
 2 2 2 2 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 6 6 6 6 6 6 6 4 4 4 4 4 4 4 4
 0 0 2 0 0 0 4 4 0 0 0 0 2 0 0 2 0 6 6 6 6 6 6 6 4 4 4 4 4 4 4 4 4 4 9 9 9
 1 1 1 1 3 1 3 3 3 3 3 3 3 3 3 6 3 6 6 3 3 6 4 6 6 1 4 6 6 4 6 6 4 1 4 6
 1 1 1 1 6 6 6 4 6 4 4 4 4 6 6 6 4 4 6 6 6 6 0 0 0 0 0 0 0 0 2 2 2 4 2 2 2 6
 2 6 6 2 2 2 2 2 6 4 6 6 6 6 4 6 6 4 4 9 9 4 5 4 4 2 2 8 8 8 8 8 8 8 8 8 8
 8 8 8 6 4 4 4 6 6 4 4 4 4 4 4 5 6 4 4 1 9 4 9 4 9 4 4 4 4 4 4 4 4 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 8 8 0 0 0 0 0 0 0 0 0
 2 2 0 0 0 0 0 2 0 0]
```

Figura 9: A matriz monstra a qual *cluster* cada dado foi alocado.

```
Localização dos centróides
[[1.36345714 1.8147619 ]
 [1.89107    2.671     ]
 [1.72423684 1.74289474]
 [2.34396216 2.48864865]
 [1.46794507 2.32211268]
 [0.78600154 2.31512821]
 [1.81406    2.2064    ]
 [3.13173333 2.55666667]
 [0.90544957 1.7973913 ]
 [1.26372235 2.72470588]]
```

Figura 10: Coordenadas dos centroides de cada *cluster* encontrado.

Dessa maneira, concluímos que o *K-means* atende a proposta da resolução desse problema. O algoritmo consegue dividir em *clusters* e classificar os dados de acordo com a necessidade apresentada. A partir dessa classificação e treinamento, torna-se possível criar um *pipeline* de dados onde essa "verificação" torna-se um "step" antes de iniciar um processo. Por exemplo, utilizando o *Apache Airflow* [13] como regente de processos de ETL, podemos encaixar o algoritmo do *K-means* como um processo de verificação antes de iniciar o ETL propriamente dito. Com isso, pode-se desencadear uma ramificação de processos de acordo com o resultado do algoritmo de ML, tornando assim o ETL mais adaptável e inteligente de acordo com o estado em que a máquina se encontra no momento.

Referências

[1] P. S. Diouf, A. Boly, and S. Ndiaye. Variety of data in the etl processes in the cloud: State of the art. In 2018 IEEE International Conference on Innovative Research and Development (ICIRD), pages 1–5, 2018.

[2] Joe Caserta and Ralph Kimball. The Data Warehouseetl Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Wiley, 2013.

[3] Simone Dominico, Eduardo Cunha de Almeida, Jorge Augusto Meira, and Marco Antonio Zanata Alves. An elastic multi-core allocation mechanism for database systems. In 2018 IEEE 34th

Apache Airflow é uma plataforma de gerenciamento de fluxo de trabalho de código aberto.

International Conference on Data Engineering (ICDE), pages 473–484. IEEE, 2018.

[4] Arpita Nagpal, Aman Jatain, and Deepti Gaur. Review based on data clustering algorithms. pages 298–303, 04 2013.

[5] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. Annals of Data Science, 2(2):165–193, 2015.

[6] Alice Zheng and Amanda Casari. Feature engineering for machine learning: principles and techniques for data scientists. "O'Reilly Media, Inc.", 2018.

[7] S. Na, L. Xumin, and G. Yong. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, pages 63–67, 2010.

[8] Malu Castellanos, Alkis Simitsis, Kevin Wilkinson, and Umeshwar Dayal. Automating the loading of business process data warehouses. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pages 612–623, 2009.

[9] Madan Gopal Devadoss, Ranveer Kumar Singh, and Panish Ramakrishna. Method for scheduling a task in a data warehouse, November 18 2014. US Patent 8,892,505.

[10] Kartick Chandra Mondal, Neepa Biswas, and Swati Saha. Role of machine learning in etl automation. In Proceedings of the 21st International Conference on Distributed Computing and Networking, pages 1–6, 2020.

[11] João Emmanuel D'Alkmin Neves, José C Conti, and Andréia R Casare. Machine learning aplicado em séries temporais em um sistema de integração de dados. Revista Brasileira em Tecnologia da Informação, 1(1):35–47, 2019.

[12] Monetdb. Monetdb: Overview. `monetdb.org/Documentation/MonetDBInternals/Overview`, 2008. Acessado em 15/04/2020.

[13] The Apache Software Foundation. Apache airflow. `https://airflow.apache.org/docs/stable/`, 2020. Acessado em 15/09/2020.