Ronaldo Nogueira de Sousa

Algoritmo Guloso para Escalonamento de Multijunções Espaciais em Sistemas Distribuídos usando o Modelo FM

Jataí-Goiás

2019

Ronaldo Nogueira de Sousa

Algoritmo Guloso para Escalonamento de Multijunções Espaciais em Sistemas Distribuídos usando o Modelo FM

Monografia apresentada ao Curso de Ciências da Computação da Universidade Federal de Goiás - Regional Jataí, como requisito parcial para obtenção do título de BACHAREL em Ciências da Computação.

Universidade Federal de Goiás - Regional Jataí - UFG-REJ

Instituto de Ciências Exatas e Tecnológicas (ICET)

Bacharelado em Ciências da Computação

Orientador: Prof. Dr. Thiago Borges de Oliveira

Jataí-Goiás

2019

Ronaldo Nogueira de Sousa
Algoritmo Guloso para Escalonamento de Multijunções Espaciais em Sistemas Distribuídos usando o Modelo FM/ Ronaldo Nogueira de Sousa. – Jataí-Goiás, 2019-
52 p. : il.

Orientador: Prof. Dr. Thiago Borges de Oliveira

Monografia (Graduação) – Universidade Federal de Goiás - Regional Jataí - UFG-REJ
Instituto de Ciências Exatas e Tecnológicas (ICET)
Bacharelado em Ciências da Computação, 2019.

1. Banco de Dados Espaciais. 2. Escalonamento de Multijunções Espaciais. 3. Algoritmo Guloso

Ronaldo Nogueira de Sousa

Algoritmo Guloso para Escalonamento de Multijunções Espaciais em Sistemas Distribuídos usando o Modelo FM

Monografia apresentada ao Curso de Ciências da Computação da Universidade Federal de Goiás - Regional Jataí, como requisito parcial para obtenção do título de BACHAREL em Ciências da Computação.

Trabalho aprovado. Jataí-Goiás, data da defesa: 12/12/2019.

Prof. Dr. Thiago Borges de Oliveira
Orientador

Prof. Dra. Joslaine Cristina Jeske de Freitas
Avaliadora

Prof. Me. Bruno Moraes Rocha
Avaliador

Jataí-Goiás
2019

Este trabalho é dedicado à minha mãe, Divalene Pereira de Sousa, que com muito carinho e apoio, nunca mediu esforços para que eu chegasse até esta etapa de minha vida.

AGRADECIMENTOS

Agradeço à minha mãe, Divalene Pereira de Sousa, que sempre me apoiou em minhas escolhas. Meu padrasto, Vicente de Oliveira Gomes, pelo apoio durante todo o meu percurso na universidade. Ao meu orientador Thiago Borges de Oliveira, pela paciência na orientação e incentivos que tornaram possível a conclusão desta monografia, além disso, pelas conversas inspiradoras que tivemos, sou muitíssimo grato.

"Nunca estou realmente satisfeita quanto a entender alguma coisa; porque, até onde entendo, a minha compreensão só pode ser uma fração infinitesimal de tudo o que eu quero compreender."

(Ada Lovelace)

RESUMO

A multijunção espacial é uma consulta importante em bancos de dados espaciais, que tem sido amplamente utilizada em muitas aplicações científicas. Por ser intensiva em dados e em computação, ela geralmente é processada em sistemas distribuídos, onde cada máquina é responsável pelo processamento de um fragmento de consulta. O fragmento de consulta é um par de partições de dados alinhadas por um predicado espacial, que chamaremos de tarefa. Para que essas tarefas sejam processadas, elas devem ser escalonadas e, nesse cenário, o objetivo principal é executar esse escalonamento de forma que a carga da consulta seja distribuída igualmente entre as máquinas, minimizando assim o *makespan*, ou seja, o tempo total de execução. Além disso, o processamento de uma tarefa exige que as partições de dados sejam transferidas para a máquina na qual a tarefa foi escalonada, ocasionando um custo de comunicação, que também deve ser minimizado. No entanto, se a partição de dados já estiver na máquina, nenhum custo de comunicação é incorrido. Para esse problema de escalonamento, dois modelos de programação linear foram propostos recentemente: o modelo FM, que considera o cenário descrito acima, e o modelo SM, que pressupõe que o custo da comunicação será incorrido toda vez que uma tarefa for processada em uma máquina. Para o modelo SM, foram desenvolvidos algoritmos que encontram soluções viáveis aproximadas. O objetivo deste trabalho foi propor um algoritmo guloso para o escalonamento de multijunções espaciais usando o modelo FM. O algoritmo guloso proposto, chamado EBCVDMC, usa o cálculo do coeficiente de variação da dispersão das tarefas para calcular o quanto a distribuição das tarefas está desbalanceada. As soluções encontradas pelo algoritmo foram melhores que as encontradas por um método de relaxação linear (LP) em 68 das 80 instâncias avaliadas. Comparado a um método de relaxação lagrangiana (LR) para o SM, o método se mostrou competitivo em relação ao valor ótimo e encontrou melhores soluções para 4 instâncias. Por ser um algoritmo de complexidade $O(nm^2)$, sendo n a quantidade de tarefas e m o número de máquinas, o EBCVDMC poderá ser usado em instâncias de multijunção para as quais o tempo de execução restrinja o tempo de escalonamento da consulta.

Palavras-chaves: *Dados Espaciais; Processamento Distribuído; Escalonamento; Algoritmo guloso.*

ABSTRACT

A multiway spatial is an important query in spatial databases, which has been widely used in many scientific applications. Because it is both data and computation-intensive, it may be processed in distributed systems, where each machine is responsible for processing a query fragment. The query fragment is a pair of data partitions aligned by a spatial predicate, which we will term as a task. For these tasks to be processed, they must be scheduled in such a way that the query load is evenly distributed across the machines, thus minimizing the makespan, or in other words, the total runtime. Moreover, processing a task requires data partitions to be transferred to the machine on which the task was scheduled, incurring a communication cost, which must also be minimized. However, if the data partition is already on the machine, no communication costs will be incurred. For this task-assignment problem, two models seek to solve it, the FM model, which considers the scenario described above, and the SM model, which assumes that the cost of communication will be incurred every time a task is processed in a machine. For the SM model, algorithms have been developed that find approximate feasible solutions. This work had as its object of study the effectiveness of the algorithms for multiway spatial scheduling. We proposed a greedy algorithm, called EBCVDMC, which uses the coefficient of variation to measure the dispersion of tasks load and to calculate how unbalanced the task distribution is. The solutions found by the algorithm were better than the ones provided by a standard linear relaxation (LP) method in 68 of 80 evaluated instances. Compared to a method based on Lagrangian Relaxation (LR), our algorithm provided competitive solutions in general and found better ones for 4 instances. As EBCVDMC complexity is $O(nm^2)$, where n is the number of tasks and m the number of machines, it can be used in small multiway spatial query instances for which runtime constrains query scheduling time.

Key-words: *Spatial Data; Distributed Processing; Scheduling; Greedy Algorithm.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SIG	Sistema de Informação Geográfica
GPS	*Global Positioning System*
DCW	*Digital Chart of the World*
LAPIG	Laboratório de Processamento de Imagens e Geoprocessamento
IBGE	Instituto Brasileiro de Geografia e Estatística
IBM	*Computer hardware company*
NASA	*National Aeronautics and Space Administration*
E/S	Entrada/Saída
CPU	*Central Processing Unit*
SHP	*Shapefile*
UPMS	*Unrelated Parallel Machine Scheduling*
UPMS-C	*Unrelated Parallel Machine Scheduling with Costs*
GB	Gigabyte
RAM	Random Access Memory
GHz	Gigahertz

LISTA DE SÍMBOLOS

θ	Predicado Espacial
$\in$	Pertence
$\bowtie$	Junção Espacial
R, S, J, A e B	Datasets
$\hat{y}_{ia}$	Variável que denota se a partição a já se encontra na máquina i
$\bar{y}_{ib}$	Variável que denota se a partição b já se encontra na máquina i
x_{ij}	Variável que denota se uma tarefa j foi escalonada para a máquina i
m	Quantidade de máquinas no *cluster*
u_i	Carga residual da máquina i
w_{ij}	Custo para processar a tarefa j na máquina i
J	*Dataset*
x_0	*Makespan*
f	Parâmetro que indica a ênfase no equilibrio do escalonamento
T	Conjunto de tarefas
M	Conjunto de máquinas
H	Hiper-grafo
V	Conjunto de vértices
N	Conjunto de arestas
cv_{wij}	Coeficiente de variação da carga de processamento gerada ao processar a tarefa j na máquina i
cv_{cij}	Coeficiente de variação do custo de comunicação gerada ao processar a tarefa j na máquina i
cv_z	Coeficiente de variação da soma do custo de processamento e do custo de comunicação

Z_{FM}^{lb}	Limite inferior para uma instância do problema para o modelo FM
Z_{FM}^{*}	Valor ótimo para para uma instância do problema para o modelo FM

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação (objeto de estudo e problema)

Nos bancos de dados espaciais, a multijunção espacial é uma importante consulta e seu crescimento tem sido impulsionado por aplicações científicas emergentes como na geografia (ex., encontrar espécies de animais que vivem em áreas de preservação ambiental que foram destruídas por queimadas) (OLIVEIRA; COSTA; RODRIGUES, 2015), na construção de placas de circuito interno (ex., encontrar circuitos lógicos que formam uma configuração topológica específica) (MAMOULIS; PAPADIAS, 2001) e no diagnóstico assistido por imagem (ex., analisar imagens tomográficas de pacientes para verificar a regressão ou não de um câncer), e estas aplicações estão cada vez mais intensivas em dados e em computação (AJI; WANG; SALTZ, 2012).

A multijunção espacial possibilita responder perguntas complexas, como, por exemplo, cruzando simultaneamente três ou mais *datasets* (CUNHA et al., 2015), além de encontrar elementos correlacionados alinhados por um predicado espacial. O seu processamento demanda uma quantidade considerável de recursos computacionais, por este motivo, pesquisas têm se concentrado na elaboração de novos métodos e técnicas para possibilitar o processamento eficiente da multijunção espacial em sistemas distribuídos (OLIVEIRA et al., 2013).

Ainda, em um sistema distribuído, cada máquina ou servidor é responsável por processar um fragmento de consulta, que iremos denominar como sendo uma tarefa constituída por um par de partições de dados de dois ou mais *datasets* alinhados por um predicado espacial. Nessa atribuição, ou escalonamento, é estabelecido o custo de comunicação e o balanceamento de carga da consulta da multijunção espacial. Aqui iremos utilizar o termo máquina como sinônimo de um servidor em um sistema distribuído.

O objetivo principal do escalonamento é realizar o escalonamento das tarefas de tal maneira que a carga de consulta seja uniformemente distribuída entre as máquinas e o custo de comunicação incorrido seja minimizado. Porém, estes são objetivos conflitantes no sentido de que para obter um balanceamento na execução da consulta incorremos em custo extra para transferir as partições de dados para máquinas ociosas. Assim sendo, que caracterizamos detalhadamente no decorrer deste texto, ainda não há uma resposta definitiva na literatura e há a necessidade de formulação de modelos que representem com maior grau de fidelidade este problema, e além disso, algoritmos que deem a solução para esses modelos (OLIVEIRA, 2017).

Há diferentes formas de como a multijunção espacial pode ser processada, chamadas

planos de execução, e a quantidade de planos não é linear quanto à quantidade de *datasets* e, por isso, é empregado um otimizador de consultas heurístico com a finalidade de escolher um bom plano de execução que reduza o consumo de recursos computacionais e o tempo de resposta da consulta. Para alcançar este objetivo é levado em consideração os metadados das partições de dados e o volume de comunicação entre os servidores em um sistema distribuído. Além disso, por exercer um impacto significativo no tempo de resposta dos algoritmos, o particionamento dos dados e a comunicação entre os servidores, em um sistema distribuído, é levado em consideração pelo otimizador de consultas (OLIVEIRA; COSTA; RODRIGUES, 2015).

A finalidade primária da fase de seleção do plano de execução de um otimizador de consulta é a seleção do plano que será usado para executar a consulta, levando em consideração o conjunto de planos possíveis e o custo de cada um. Já que a quantidade de planos é exponencial em relação ao número de *datasets*, o tempo para selecionar um plano dever ser considerado da perspectiva do tempo total de execução da consulta e da taxa de transferência do sistema, sendo esta uma fase crítica do otimizador de consultas que há no escalonamento do plano de consulta, ou seja, no algoritmo de seleção do plano (OLIVEIRA, 2017).

Além de selecionar um bom plano de execução, em um sistema distribuído o otimizador de consultas deve especificar em qual servidor ou máquina o fragmento da consulta deverá ser executada. Por conseguinte, esta decisão implica que deverá também ser especificado de onde a partição de dados deverá ser copiada, observando que as partições de dados são distribuídas a priori (OLIVEIRA, 2017). Um bom escalonamento faz com que as máquinas fiquem ocupadas na mesma quantidade de tempo (execução balanceada) e que haja a menor quantidade de transferência de dados possível entre as mesmas.

Na tese de Oliveira (2017), definiram-se dois modelos para este problema de otimização: o modelo FM e SM. O modelo FM busca minimizar tanto o custo de processamento quanto o de comunicação. O modelo SM é uma simplificação do modelo FM e para ele foram definidos algoritmos de otimização combinatória. Contudo, estes ignoram o fato de que a partição de dados não deve ser transferida de forma redundante pela rede, assumindo que ela será copiada todas as vezes em que for usada na máquina. Devido a esta simplificação do modelo SM, uma desvantagem é que há uma perda de oportunidades na redução do custo de comunicação em detrimento da melhora do custo de carga de consulta. Vale destacar que uma solução para o modelo SM é uma solução viável para o modelo FM sem contudo ser uma solução ótima (OLIVEIRA, 2017).

A otimização desse escalonamento é uma extensão do problema de escalonamento com custo de máquinas não relacionadas (*the unrelated parallel machines scheduling problem with costs*) e é um problema NP-*hard*. Logo, a existência de um algoritmo de tempo polinomial para o FM é bem remota. Foi investigado por Oliveira (2017) o quão

rápido e o quão bem o ótimo para o modelo FM pode ser aproximado explorando métodos que resolvessem sua versão simplificada (SM). Para isso, foi aplicado exaustivamente algoritmo de *branch-and-bound* e foi observado pelo autor que algumas vezes houve diferenças significativas entre os dois escalonamentos. Portanto, isso implica que há espaço para melhorar ainda mais o processo de escalonamento conforme afirmado pelo autor. Deste modo, o desenvolvimento de métodos aproximados para resolver o modelo FM devem ser considerados.

Portanto, o tema deste trabalho é o escalonamento da multijunção espacial em *cluster* de computadores, sendo o objeto de estudo a efetividade dos algoritmos para escalonamento de multijunções espaciais.

1.2 Objetivo do Trabalho

A existência de um algoritmo em tempo polinomial para o FM é bem remota, e é sabido que o método guloso não é eficiente quando comparado com os métodos de otimizações tradicionais. Porém, este projeto buscou propor um método que consiga um bom desempenho em termo de execução e resultados heurísticos aceitáveis.

Com base na motivação apresentada na Seção 1.1, o objetivo geral deste trabalho foi propor um algoritmo com o paradigma guloso para escalonamento de multijunções espaciais empregando o modelo FM. Para atingir este objetivo tem-se os seguintes objetivos específicos:

- Estudar o modelo SM e FM;
- Identificar as propriedades do problema que podem ser explorados de forma gulosa;
- Propor um algoritmo guloso para o modelo FM;
- Realizar um estudo comparativo entre o novo algoritmo proposto e os existentes para o modelo SM.

1.3 Contribuição do Trabalho

A principal contribuição deste trabalho foi o desenvolvimento de um algoritmo guloso para o modelo FM. O algoritmo proposto possui complexidade $O(nm^2)$, o que o torna um bom candidato para escalonar instâncias de multijunção que possuem um tempo de execução pequeno, o qual restringe o tempo disponível para o escalonamento da consulta.

1.4 Organização da Monografia

Este trabalho está dividido em seis capítulos, descritos resumidamente a seguir: O segundo capítulo aborda conceitos fundamentais para entendimento do trabalho. O terceiro capítulo apresenta os trabalhos relacionados. O quarto capítulo apresenta os dois algoritmos que foram desenvolvidos. O quinto capítulo apresenta os resultados que foram obtidos e também discorre sobre eles. Já no sexto capítulo, é apresentado as conclusões finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Introdução

Neste capítulo serão apresentados alguns conceitos introdutórios para uma melhor compreensão desta monografia. Assim sendo, as próximas seções abordarão uma breve introdução sobre dados espaciais, sistema de informação geográfica, consultas espaciais, processamento distribuído de junção espacial e o modelo FM.

2.2 Dados Espaciais

Os dados espaciais referem-se aos objetos geográficos do mundo real de interesse, como ruas, prédios, lagos e países, e suas respectivas localizações. Estes objetos possuem certos atributos de interesse, como por exemplo: nome, número de histórias, profundidade, ou população (CAMPBELL, 2012). Além disso, estes dados podem ser representados espacialmente, ou seja, de forma gráfica, e estes constituem-se em imagens, mapas temáticos ou planos de informações. Para tais tipos de dados a estrutura básica pode ser vetorial ou matricial (FITZ, 2008).

A estrutura matricial, ilustrada na Figura 1, é composta por linhas e colunas de pixels de tamanhos iguais interconectados para formar uma superfície plana e utilizados para formar pontos, linhas, áreas, redes e superfícies. Ao empregar este tipo de estrutura deve-se tomar cuidado ao determinar a resolução da mesma, pois ao utilizar uma resolução excessivamente grosseira de pixel resultará em perda de informações, contudo, usando uma resolução excessivamente boa resultará no aumento significativo do tamanho dos arquivos e, por conseguinte, nos requisitos de processamento do computador para exibição e análise (CAMPBELL, 2012).

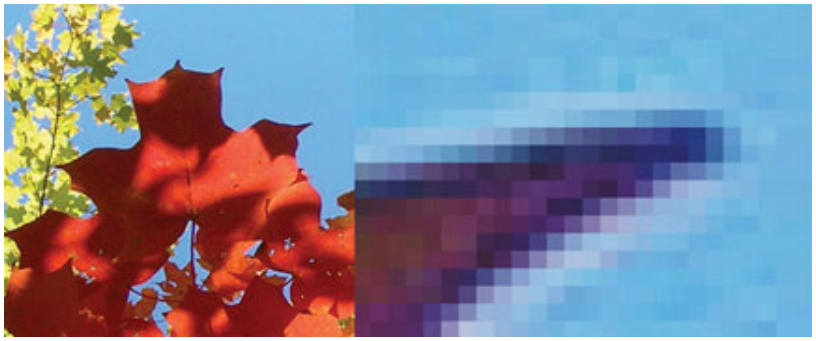


Figura 1 – Estrutura de dados matricial. Retirado de (CAMPBELL, 2012)

Em contraste à estrutura matricial há a vetorial, ilustrada na Figura 2. Neste

modelo, o espaço não é quantificado em uma grade discreta de células como na estrutura anterior. A estrutura de dados vetoriais usam pontos e seus pares X, Y associados para representar os vértices de características espaciais. Diferentemente da estrutura matricial, o modelo vetorial tende a representar melhor a realidade devido a precisão dos pontos, linhas e polígonos. Além disso, esta estrutura tende a ser mais compacta, logo, o tamanho dos aquivos são tipicamente menores do que comparados com a estrutura matricial (CAMPBELL, 2012). Neste projeto a estrutura de dados espacial considerada será a vetorial.

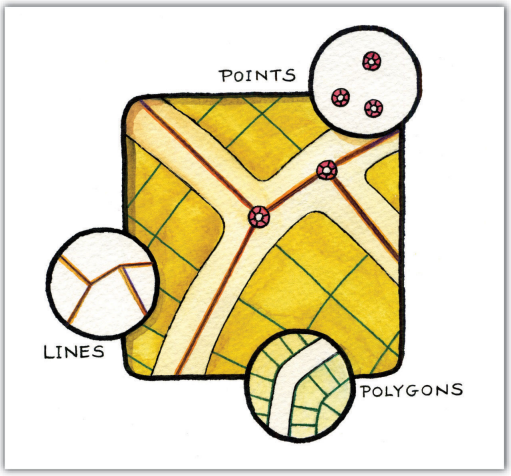


Figura 2 – Estrutura de dados vetorial. Retirado de (CAMPBELL, 2012)

Independente da estrutura de dados espacial, seja matricial ou vetorial, ela deve está vinculada a dados ditos tradicionais, dados alfanuméricos. Em um Sistema de informação geográfica (SIG), em geral, é preferível o uso de estrutura vetorial para a conexão desses dados (FITZ, 2008). Na próxima seção será abordado acerca do SIG e sua relação com dados espaciais.

2.3 Sistema de Informação Geográfica

Para CAMPBELL (2012), consiste em um tipo especial de programa capaz de armazenar, editar, processar, apresentar dados geográficos e informações como mapas. Além disso, o SIG é composto por um sistema de gerenciamento de banco de dados capaz de manipular e integrar dois tipos de dados: dados espaciais e dados de atributos.

O SIG é mais que uma ferramenta cartográfica para a produção de mapas, uma vez que é capaz de armazenar, recuperar e combinar os dados para criar uma nova representação do espaço geográfico, possibilitando assim análises geográficas e a execução de simulações de forma a ajudar usuários especialistas a organizar seu trabalho em várias áreas. Como por exemplo, na administração pública, redes de transporte, aplicações militares e em sistemas de informação ambiental (RIGAUX; SCHOLL; VOISARD, 2002).

As informações geoespaciais correspondentes a um tópico específico são organizadas em um tema e este é similar a uma relação definida em um banco de dados relacional. Os rios, cidades e países são exemplos de temas. Um tema é uma coleção de objetos geográficos que correspondem a uma entidade do mundo real contendo uma descrição constituída por atributos descritivos, por exemplo, o nome e a população de um país, e um componente espacial do qual pode englobar tanto a geometria quanto a topologia desta entidade (CAMPBELL, 2012).

A representação da geometria e da topologia requer uma modelagem poderosa a nível de tema ou objeto, o que leva a modelos de dados espaciais. Geralmente, os tipos básicos de dados em um modelo de dados espaciais são: ponto (objeto zero-dimensional), linha (unidimensional) e região (objeto 2D). Por exemplo, o objeto espacial associado a um rio é uma linha, enquanto que o objeto associado a uma cidade é uma região (polígono). Uma terceira, e até mesmo uma quarta, dimensão pode ser introduzida se levar em consideração o volume ou o tempo (CAMPBELL, 2012).

Em geral os SIGs separam os dados em camadas de informações, ou temas, denominadas de *layers* e estas camadas podem ser livremente manipuladas gerando informações adicionais às preexistentes. O procedimento mais importante para um SIG é a capacidade de executar a sobreposição de camadas de dados espaciais, conhecida como *overlay* (FITZ, 2008), que é o processo de formar um novo mapa a partir da sobreposição de dois ou mais mapas temáticos diferentes da mesma área (CAMPBELL, 2012).

O princípio da organização dos dados em um SIG é a camada de informação que é uma representação tanto de áreas discretas quanto contínuas, ou de uma coleção de objetos do mesmo tipo. Usualmente, os dados são organizados de forma a manter os elementos similares em um único *layer*, por exemplo, todos os pontos telefônicos ficariam em um *layer*, e todas as linhas rodoviárias em outro. Uma camada de informação contém tanto dados espaciais quanto dados de atributos que descrevem a área ou os objetos no *layer* (HUISMAN; BY, 2009).

A parte mais essencial de um SIG é a sua função de análise espacial, como por exemplo, operadores que usam dados espaciais para obter uma nova informação. As consultas espaciais desempenham um papel importante nesta função. Elas processam os dados espaciais e de atributos em um banco de dados espacial para obter informações a partir deles (HUISMAN; BY, 2009).

2.4 Consultas espaciais

A análise espacial é o componente fundamental de um SIG que permite um estudo profundo das propriedades topológicas e geométricas de um ou mais *dataset*. A exibição seletiva e a recuperação de informações compõe partes essenciais de qualquer sistema de

informações geográficas. As consultas espaciais estão inseridas neste contexto, uma vez que com elas é possível responder questões colocadas ao banco de dados (CAMPBELL, 2012).

Consultas espaciais são as operações fundamentais empregadas na análise espacial que é a manipulação de dados espaciais com o propósito de adicionar valor ao mesmo, ou seja, apoiando decisões ou até mesmo revelando informações que não estavam aparentes. O três tipos importantes de consultas espaciais são (RIGAUX; SCHOLL; VOISARD, 2002):

- *Window query*: É uma seleção aplicada a um único *dataset* possuindo como resultado um conjunto de objetos que sobrepõe uma dada região, geralmente retangular;
- *Spatial Join*: Uma operação de sobreposição (*overlay*) aplicado a dois *datasets*. Neste tipo de consulta espacial, o objeto espacial de um *dataset* é *joined* com um outro objeto de outro *dataset* se suas geometrias satisfazem a um predicado espacial, tal como a intersecção; e
- *Multiway Spatial Join*: É um *Spatial Join* que envolve um número arbitrário de *datasets* com cada par de *datasets* possuindo um predicado espacial particular.

Dado dois *datasets* de objetos multidimensionais no espaço Euclidiano, uma junção espacial encontra todos os pares de objetos que satisfazem uma dada relação entre os objetos que envolvem os valores de seus componentes espaciais, como a intersecção. Por exemplo, uma junção espacial responde perguntas como encontrar todas as áreas rurais que estão abaixo do nível do mar, dado um mapa de elevação e um mapa de uso da terra (JACOX; SAMET, 2007).

Formalmente a junção espacial pode ser definida como segue: dado dois *datasets* R e S e um predicado de junção espacial θ (por exemplo: proximidade, sobreposição, intersecção) encontre todos os pares de objetos (r, s) onde $r \in R$ e $s \in S$ no qual o predicado θ é satisfeito para todo (r, s). Podem haver mais de dois *datasets* na relação (multijunção espacial) ou somente um e os *datasets* podem ter mais do que duas dimensões (JACOX; SAMET, 2007).

Uma junção espacial responde uma questão aplicando um algoritmo geométrico para a verificação do predicado espacial em cada par de atributos espaciais e para uma junção espacial há várias opções de predicados espaciais. A seguir, serão descritos alguns deles (CAMPBELL, 2012):

- Intersecção: Seleciona todos as características na camada de destino que compartilham uma localidade comum com a camada de origem e permite que camadas de ponto, linhas ou polígonos sejam usadas como as camadas de origem e destino;

- Estão dentro de um distância: Neste tipo de consulta é especificado uma valor para a distância que é usado para armazenar em *buffer* a camada de origem. Todas as características que intersecta este *buffer* é destacado na camada de destino. Esta consulta permite que a camada de pontos, linhas ou polígonos sejam ambos a camada de origem e destino;

- Completamente contém: Esta técnica de consulta espacial retorna todas as características que estão completamente contidas e que não possua os limites coincidentes na camada de origem. Esta consulta permite pontos, linhas e polígonos como sendo a camada de origem, mas somente polígonos podem ser usados como a camada de destino.

- Completamente dentro: Seleciona todas as características na camada de destino em que a extensão espacial inteira ocorre dentro da geometria da camada de origem. Esta consulta permite que pontos, linhas, ou polígonos sejam a camada de destino, porém somente polígonos podem ser usados como a camada de origem.

- Tenha seu centro em: Seleciona as características de destino cujo centro ou centroide, estão localizadas dentro do limite do conjunto de dados da característica de origem permitindo tanto a origem quanto o destino serem ambos pontos, linhas ou polígonos.

A multijunção espacial (*Multiway Spatial Join*) é um tipo de junção espacial, porém, com um número arbitrário de *datasets*. Um exemplo de consulta de multijunção espacial é: *"Encontre todas as espécies de animais que vivem em áreas de preservação que foram danificadas pelo fogo na margem de um rio"*. Para este tipo de questão seriam utilizadas quatro *datasets* combinados para computar o resultado para esta consulta, sendo eles: animais, áreas de preservação, propagação de fogo e rios (OLIVEIRA, 2017).

A computação de tarefas espaciais como a junção espacial são intensivas tanto em dados quanto em computação (PURI; PAUDEL; PRASAD, 2018) e uma vez que há uma maior disponibilidade de dados espaciais em larga escala, muitos algoritmos para processar a junção espacial de forma paralela ou distribuída foram propostos (OLIVEIRA, 2017).

2.5 Processamento Distribuído de Junção espacial

Tecnologias como satélites, dispositivos médicos e dispositivos habilitados para GPS tem possibilitado o aumento na produção de dados espaciais. Por exemplo, os arquivos de dados de satélites da *National Aeronautics and Space Administration (NASA)* ultrapassou 500 TB e continua crescendo (PURI; PAUDEL; PRASAD, 2018).

Algumas aplicações incluem simulação de incêndios florestais ou furacões, e, para tal, a junção espacial se faz necessária para prever áreas afetadas e resgatar abrigos.

Estes tipos de aplicações exigem o aproveitamento de técnicas de computação de forma a aproveitar bem os recursos computacionais para produzir resultados em tempo real (PRASAD et al., 2017).

Por ser intensiva em dados e em computação, mais e mais pessoas buscam utilizar do processamento distribuído ou paralelo para o processamento da consulta de junção espacial (YU; WU; SARWAT, 2015) com o objetivo de acelerar o processamento das consultas espaciais, ou seja, distribuindo a carga da consulta uniformemente entre os servidores (OLIVEIRA, 2017). Neste contexto há alguns trabalhos como Hadoop-GIS, SpatialHadoop, Sphinx e MSJS, porém, como discutido em Oliveira (2017), este sistemas possuem alguns problemas, sendo alguns deles respectivamente:

- o otimizador de consultas é baseado em regras e não em custos;
- dá suporte somente a junção espacial simples e não há uma estratégia para processar a multijunção espacial;
- embora tenha um plano de execução distribuído antes da execução da consulta, tem como objetivo reduzir o custo de comunicação no *cluster*;
- não há uma estratégia para determinar os custos da consulta.

Geralmente o processamento de uma multijunção espacial é significativamente mais complexo do que o processamento de uma junção espacial pois ela pode ser executada de muitas maneiras diferentes, chamadas de planos de execução. Com isso, surgiram propostas de otimizadores de consulta baseado em custo para selecionar um bom plano de execução para uma consulta com base nos seus custos computacionais (OLIVEIRA, 2017).

O otimizador de consultas pode ser baseado em regras ou baseado em custo. O otimizador baseado em regras usa uma regra fixa para selecionar um bom plano de execução, embora seja simples e rápido para selecionar o plano, é menos flexível no que diz respeito a adição de novos algoritmos de junção. Além disso, este otimizador têm sensibilidade limitada às propriedades do conjunto de dados que não foram consideradas em seu *design* (FORNARI; COMBA; IOCHPE, 2007).

Em contraste, o otimizador baseado em custo estima um custo para cada plano de execução, tendo como base as propriedades do *dataset*, assim como o custo de E/S e *Central Processing Unit (CPU)* dos algoritmos de junção. Evidentemente é necessário estimar o custo de muitos planos de execução, além de precisar coletar os metadados dos *datasets* a priori (OLIVEIRA, 2017).

A maioria dos algoritmos para processar a junção espacial de forma paralela ou distribuída utiliza a estratégia de particionamento de dados para dividir os objetos dos *datasets* em grupos, chamados de partições de dados ou células (OLIVEIRA, 2017).

Tendo em vista que os dados espaciais não são uniformes por natureza, a transferência de dados é de difícil otimização, e quando distribuídos pelo *cluster* geram partições de tamanhos não uniformes. Isso complica o processamento pois partições de dados de tamanhos diferentes também possuem tempos diferentes de processamento, além do problema de transferência (PURI; PAUDEL; PRASAD, 2018).

Neste contexto, o processamento distribuído da junção espacial deve escalonar as partições de dados para a máquina na qual serão processadas e este pode ser visto como um problema de atribuição de tarefas onde busca-se escaloná-las a um conjunto de servidores em um sistema distribuído. Cada tarefa é constituída por um par de partições de dados que estão alinhadas a um predicado espacial para uma junção ou multijunção espacial (OLIVEIRA, 2017).

O objetivo principal do escalonamento é alocar as tarefas de forma que a carga da consulta seja distribuída uniformemente entre os servidores e o custo de comunicação incorrido seja minimizado (OLIVEIRA, 2017). Porém, como já dito, os dados espaciais não são uniformes por natureza e naturalmente as partições de dados também não são uniformes.

2.6 Modelo FM

Na tese de Oliveira (2017) foi definido o modelo FM. No modelo FM é levado em consideração o custo de comunicação incorrido ao transferir uma partição de dados de uma máquina, onde ela está localizada, para a máquina na qual ela foi escalonada para ser processada, sendo que nenhum custo de comunicação é incorrido caso a partição de dados for processada na máquina em que ela esteja localizada.

O custo de processamento é definido baseado na quantidade de tempo de processamento necessário para realizar a tarefa. Além disso é considerada uma carga residual para a máquina, que surge da execução de uma consulta anterior ou alguma outra particularidade do sistema. Neste contexto, o modelo busca minimizar tanto o custo de processamento quanto o custo de comunicação.

Formalmente, dado dois *datasets* A e B utilizados em uma multijunção espacial, e seja $p = |A|$ e $q = |B|$, e m o número de máquinas, onde $m \leq min(p, q)$, o conjunto de *jobs* J a serem processados é composto de pares de objetos $\{a, b\}$, onde $a \in A, b \in B, J \subseteq A \times B$, e $n = |J|$ e para cada *job* $j \in J$ possui um peso associado, $w_j \in \mathbb{Z}$, que especifica o custo de processamento. Para cada a e b há um custo associado, $\hat{c}_{ia} \in \mathbb{Z}^+$ e $\bar{c}_{ib} \in \mathbb{Z}^+$, respectivamente, o qual especifica o custo de comunicação quando o *job* é processado na máquina $i, 1 \leq i \leq m$. Além disso, cada máquina possui uma carga residual que reduz a sua capacidade de processamento denotada no modelo por u_i, e o parâmetro f indica a ênfase desejada do quão equilibrado deve ser o escalonamento.

O modelo possui três conjuntos de variáveis de decisão binárias:

- $\hat{y}_{ia} = 1$, se a for processada na máquina i, do contrário $\hat{y}_{ia} = 0$, $\forall\ a \in A, i = 1, ..., m$;
- $\bar{y}_{ib} = 1$, se b for processada na máquina i, do contrário $\bar{y}_{ib} = 0$, $\forall\ b \in B, i = 1, ..., m$;
- $x_{ij} = 1$, se o *job* for processado na máquina i, do contrário $x_{ij} = 0$, $\forall\ j \in J, i = 1, ..., m$.

Além disso, seja x_0 uma variável de decisão que representa o tempo de conclusão do último *job* processado por qualquer máquina, por exemplo, o *makespan*. O modelo FM é definido a seguir nas equações FM.1 até FM.6:

$$Z_{FM} = \text{Min } fx_0 + \sum_{i=1}^{m} \left(\sum_{a=1}^{p} \hat{c}_{ia}\hat{y}_{ia} + \sum_{b=1}^{q} \bar{c}_{ib}\bar{y}_{ib} \right), \tag{FM.1}$$

$$Sujeito\ a: \sum_{i=1}^{m} x_{ij} = 1, \qquad \forall\ j \in J \tag{FM.2}$$

$$\sum_{j \in J} w_j x_{ij} + u_i \leq x_0, \qquad i = 1, \ldots, m \tag{FM.3}$$

$$x_{ij} - \hat{y}_{ia} \leq 0, \qquad \forall\ j = \{a, b\} \in J,\ i = 1, \ldots, m \tag{FM.4}$$

$$x_{ij} - \bar{y}_{ib} \leq 0, \qquad \forall\ j = \{a, b\} \in J,\ i = 1, \ldots, m \tag{FM.5}$$

$$x_{ij},\ \hat{y}_{ia},\ \bar{y}_{ib} \in \{0, 1\}. \qquad \forall\ j = \{a, b\} \in J,\ i = 1, \ldots, m \tag{FM.6}$$

A função FM.1 representa o objetivo de minimizar o tempo de processamento e a soma dos custos de comunicação. A família de restrições em FM.2 expressa o requisito de que apenas um *job* deve ser processado em exatamente uma máquina. A família de restrições em FM.3 é o conjunto de inequações lógicas que surge da necessidade de minimizar o tempo de processamento. Já a família de restrições em FM.4 e FM.5 é um conjunto de inequações lógicas que indicam se uma partição de dados é ou não processada por uma máquina específica. A família de restrições em FM.6 representa as restrições usuais de integralidade, ou seja, indicando se um *job* é ou não processado por uma máquina (x_{ij}), e se uma partição de dados é ou não usada por uma máquina ($\hat{y}_{ia}, \bar{y}_{ib}$).

Este problema é uma extensão do problema de escalonamento de máquinas paralelas não relacionadas com custos e este é um problema *NP-Hard*. Portanto, a menos que N=NP, a existência de um algoritmo de tempo polinomial para resolver FM é remota. Além disso, ainda não foi apresentado uma solução para o modelo FM, contudo foi apresentado uma simplificação para o mesmo.

Com intuito de simplificar o modelo FM, (OLIVEIRA, 2017) propôs o modelo SM que ignora o fato de que a partição de dados não deve ser transferida de maneira

redundante pela rede, assumindo que ela sempre será copiada toda vez em que for usada no servidor.

A desvantagem do modelo SM é que o escalonamento resultante pode gerar perda de oportunidades para reduzir o custo geral da comunicação, devido a concentração de tarefas envolvendo a mesma partição em um determinado servidor.

Foi aplicado exaustivamente, sempre que foi possível, ao modelo FM algoritmos de *branch-and-bound* com intuito de comparar cada escalonamento do modelo SM quando considerada uma instância do modelo FM. Além disso, Oliveira (2017) descreve um exemplo de um caso real de uma consulta que pode ter uma redução de cerca de 20% do custo de comunicação utilizando-se o modelo FM ao invés do modelo SM para escalonar a mesma. Posto isto, chegou-se a conclusão de que há espaço para melhorar ainda mais o processo de escalonamento.

2.7 Algoritmos Gulosos

O paradigma guloso é tido, na ciência da computação, como uma técnica de projeto de algoritmo aplicada a problemas de otimização. Nesta técnica, a construção da solução para o problema ocorre em uma sequencia de passos, cada um expandindo uma solução parcial obtida até então, até que uma solução completa seja construída (LEVITIN, 2012).

A essência desta técnica consiste no fato de que as decisões tomadas localmente devem respeitar as restrições do problema, serem a melhor escolha dentre todas as outras na etapa atual e uma vez tomadas não podem ser desfeitas. Para alguns problemas, esta sequencia de decisões locais resultam em soluções ótimas. Para outros, no entanto, este não é o caso. Mesmo assim, para tais problemas, um algoritmo guloso ainda tem valor se há interesse em soluções aproximadas (LEVITIN, 2012).

Na tese de Oliveira (2017) a estratégia gulosa foi utilizada com dois propósitos: comparar o desempenho dos métodos combinatórios, LP e LR, com o desempenho de um método mais simples e para usá-lo quando o limite de tempo imposto à otimização de consultas for crítico, por exemplo, para consultas com pequeno tempo de execução. Porém, a estratégia gulosa proposta foi para o método SM, enquanto que as estratégias que aqui são propostas resolvem o modelo FM.

3 TRABALHOS RELACIONADOS

3.1 Introdução

Neste capítulo são apresentados os trabalhos mais relacionados ao escalonamento de tarefas em sistemas distribuídos. Os critérios de levantamento bibliográfico são descritos na Seção 3.2, os trabalhos são discutidos na Seção 3.3 e uma tabela comparativa é apresentada na Seção 3.4, ilustrando as diferenças comparadas com esta proposta.

3.2 Critérios de busca

Por serem problemas bem caracterizados e estudados na literatura, e que apesar de não tratarem especificamente de dados espaciais ou sistemas distribuídos, abordam o escalonamento de tarefas em máquinas não relacionados de forma geral, e, por esse motivo, foi escolhido como trabalhos relacionados o problema UPMS e UPMS-C.

Além disso, o trabalho *Efficient processing of multiway spatial join queries in distributed systems* foi selecionado pelo fato de que os resultados foram comparados com os obtidos por este trabalho. O ultimo trabalho escolhido se deve a fato de que ele também busca escalonar o processamento de dados espaciais, porém, utilizando hiper-grafo.

3.3 Trabalhos analisados

3.3.1 An optimal rounding gives a better approximation for scheduling unrelated machines (T1)

O problema UPMS, é definido como sendo: dado um conjunto T de tarefas, um conjunto M de máquinas, e para cada $t \in T$ e $i \in M$, $p_{it} \in Z^+$ o tempo para processar a tarefa t na máquina i, escalone as tarefas de forma a minimizar o *makespan*, ou seja, o tempo total de execução (VAZIRANI, 2001).

Shchepin e Vakhania (2005) apresentam algoritmo aproximado para o problema UPMS. Na abordagem empregada, a taxa de desempenho do escalonamento é a razão entre o *makespan* gerado pelo escalonamento e o *makespan* ideal. Um algoritmo com a taxa de desempenho no pior caso d é chamado de algoritmo d-aproximação. Neste sentido, Shchepin e Vakhania (2005) apresentam um algoritmo $(2 - 1/m)$-aproximação, onde m é a quantidade de máquinas, e provaram que esta é a melhor taxa de aproximação para esta abordagem.

O modelo FM se difere deste trabalho no sentido que ele busca minimizar os custos na função objetivo e também o *makespan*. Além disso, são adicionadas restrições para computar o custo total, restrições FM.4 e FM.5.

3.3.2 A FPTAS for approximating the unrelated parallel machines scheduling problem with costs (T2)

Neste trabalho os autores Angel, Bampis e Kononov (2001) propõem um algoritmo aproximado para resolver o problema UPMS-C onde um conjunto de n tarefas devem ser processadas em m máquinas e cada tarefa deve ser processada somente para uma máquina. Neste cenário, processar a tarefa j na máquina i requer um tempo $p_{ij} \geq 0$ além de um custo c_{ij}. O objetivo é encontrar um escalonamento de forma que se obtenha um balanceamento entre o *makespan* e o custo total.

Embora haja uma certa similaridade deste problema com o modelo FM, em relação a função objetivo, eles são diferentes, pois o FM apresenta restrições adicionais FM.4 e FM.5 que são necessárias para calcular o custo total, usando cada partição de dados que compõe uma tarefa.

3.3.3 Hypergraph+: An improved hypergraph-based task-scheduling algorithm for massive spatial data processing on master-slave platforms (T3)

Cheng et al. (2016) propõem um algoritmo baseado em hiper-grafo para escalonamento de tarefas para o processamento de dados espaciais massivo em plataforma mestre/escravo. Neste contexto, uma tarefa para ser executada depende de um subconjunto de arquivos. O conjunto de arquivos está inicialmente armazenado em um servidor mestre, ou seja, se uma tarefa precisar de um arquivo e este não está no servidor escravo ao qual ela foi escalonada, então, este arquivo deverá ser requisitado do servidor mestre, além disso, dado uma arquivo, este pode ser compartilhado entre várias tarefas.

Um hiper-grafo H = (V, N) é definido como sendo um conjunto de vértices V e um conjunto de hiper-arestas N que conecta os vértices. Cada hiper-aresta é um subconjunto não vazio de vértices. Nesse cenário, os vértices representam as tarefas e as arestas representam os arquivos. Uma hiper-aresta conectada a alguns vértices significa que estes vértices compartilham este arquivo. Os vértices possuem pesos que indicam o tempo necessário para processá-lo e as hiper-arestas possuem um pesos equivalente ao tamanho do arquivo, indicando o custo de comunicação. O tempo estimado para o processamento de uma tarefa é a soma do peso da hiper-aresta e o tempo necessário para processar a tarefa.

A estratégia utilizada é a criação de partições a partir do hiper-grafo, o que naturalmente resultará em partições ligadas por hiper-arestas. Nessa circunstância, o

objetivo do particionamento é minimizar a quantidade de cortes necessários para separar estas partições. Após feito as partições, elas serão escalonadas para os servidores escravos.

No cenário descrito, este trabalho é similar ao modelo FM no que diz respeito a utilização de arquivos que podem ser vistos como sendo as partições que compõem a tarefa. O fato dos arquivos serem compartilhados na máquina também demonstra esta similaridade, porém a forma como é computada o custo é diferente, enquanto no FM tem restrições necessárias para calcular o custo, o Hypergraph+ não as possuem mas emprega o particionamento do grafo de forma a minimizar o custo total de comunicação. Outra diferença é que o Hypergraph+ calcula o tempo para processar a tarefa como sendo o tempo para processá-la mais o tempo para transferir os arquivos que a tarefa precisa para ser processada.

3.3.4 Efficient Processing of Multiway Spatial Join Queries in Distributed Systems (T4)

A tese de Oliveira (2017) propõe o modelo FM que modela o escalonamento de fragmentos de consultas de multijunção espacial. Como o modelo se trata de um problema NP-Hard, o autor cria um modelo simplificado para o FM, denominado de modelo SM.

Neste cenário, uma tarefa é um par de partição de dados alinhado a um predicado espacial. O modelo FM leva em consideração o fato de que uma partição de dados será copiada uma única vez para cada máquina na qual ela será processada e nenhum custo de comunicação é incorrido se ela é processada na máquina na qual ela já se encontra. A sua simplificação, ou seja, o modelo SM, assume que será copiada cada vez que for usada em uma tarefa, mesmo que as tarefas distintas que a empregam sejam alocadas na mesma máquina.

Uma solução viável para o modelo SM é também uma solução factível para o modelo FM, com a diferença de que o modelo FM leva em consideração de que a partição de dados não deve ser transferida de forma redundante pela rede (OLIVEIRA, 2017).

Neste contexto foram desenvolvidos algoritmos que resolvessem o modelo SM e, conforme demonstrado pelo autor, há espaço para melhorar ainda mais o processo de escalonamento, visto que, houve diferenças significativas entre os resultados para o SM e para o modelo FM, sendo que para obtenção dos resultados para o modelo FM foi empregado algoritmo de *branch-and-bound*.

3.4 Resumo Comparativo

A partir da leitura dos trabalhos relacionados foi possível notar algumas diferenças neles em relação a este trabalho como pode ser observado na Tabela 1. Os critérios de

diferenciação são:

- C1: Formaliza o problema em um modelo de programação linear
- C2: Paradigma do algoritmo para escalonamento é guloso
- C3: Considera o custo de *makespan* e de comunicação

O primeiro critério foi escolhido pois, o modelo de programação linear é constituída por uma função objetiva, onde buscamos minimizá-la ou maximizá-la, sujeita a um conjunto de restrições expressadas como inequações lineares, e nesse cenário, o problema que buscamos resolver é melhor expressado por este modelo, por exemplo, a restrição de que uma tarefa somente pode ser processada por uma máquina. Já o segundo critério foi escolhido pelo fato de que neste trabalho buscamos resolver o problema utilizando o paradigma guloso. Por fim, o terceiro critério se deve ao fato de o problema que buscamos resolver tem como objetivo minimizar o *makespan* e o custo de comunicação.

Tabela 1 – Comparativo entre trabalhos.

Trabalhos	Modelo de Programação Linear	Algoritmo Guloso	*Makespan* e Comunicação
UPMS (T1)	Sim	Não	Não
UPMS-C (T2)	Sim	Não	Não
Hypergraph+ (T3)	Não	Sim	Sim
LP (T4)	Sim	Não	Sim
LR (T4)	Sim	Não	Sim
Essa proposta	Sim	Sim	Sim

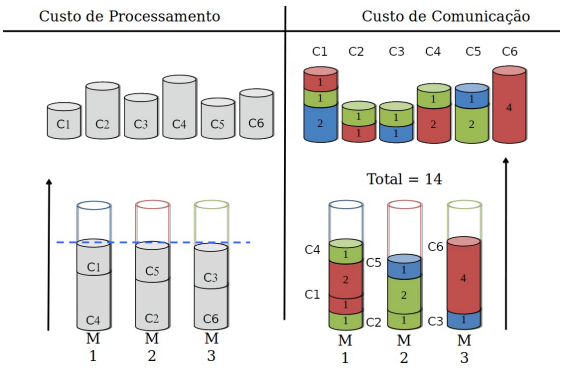
4 ALGORITMOS GULOSOS PARA O MODELO FM

4.1 Introdução

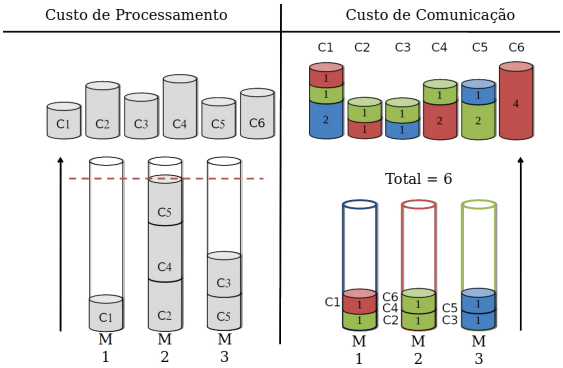
Neste capítulo são apresentados os dois algoritmos gulosos desenvolvidos para o escalonamento de tarefas. Na Seção 4.2 são apresentadas duas estratégias gulosas. Na Seção 4.3 é apresentado o algoritmo EBCVD. Já na Seção 4.4 é apresentado o algoritmo EBCVDMC. E por último, na seção 4.5, é apresentado o algoritmo utilizado para calcular o coeficiente de variação.

4.2 Estratégias gulosas para o modelo FM

A forma como as tarefas são distribuídas pelo *cluster* influencia o quanto será o *makespan*, por exemplo, se colocarmos todas as tarefas em uma única máquina teremos um *makespan* alto, todavia, teremos um custo de comunicação baixo, pois, partições repetidas seriam copiadas uma única vez. Existe ainda mais uma situação, se escalonarmos as tarefas somente pela perspectiva de tornar o *makespan* o menor possível, o escalonamento resultará em um custo de comunicação elevado, o inverso disso também é verdadeiro. A Figura 3 ilustra este comportamento, onde a linha tracejada indica o valor do *makespan*.



(a) Escalonamento com o menor *makespan*, linha azul



(b) Escalonamento com o menor custo de comunicação, mas com *makespan*, linha vermelha, alto

Figura 3 – Ilustração de dois escalonamentos sob perspectivas diferentes. Em a) as decisões tomadas levaram a um *makespan* menor, já em b) as decisões tomadas levaram a um menor custo de comunicação.

Refletindo sobre o comportamento ilustrado na Figura 3, surgiu a primeira estratégia, que, busca tomar decisões, em cada passo do algoritmo, que tornem a distribuição

das tarefas balanceadas ao passo que o custo de comunicação também fique balanceado. Para isso, foi utilizado o coeficiente de variação que nos dá a qualidade da distribuição das tarefas, onde quanto mais próximo de zero melhor o balanceamento. Desse modo, o desbalanceamento resulta na soma do coeficiente de variação da carga de processamento e do custo de comunicação.

Assim sendo, o coeficiente de variação para a carga de processamento é definido como sendo a razão entre o desvio padrão e a média da mesma. O mesmo processo é aplicado para o custo de comunicação. Neste cenário, em cada etapa o algoritmo calcula o desbalanceamento gerado ao escalonar uma tarefa para uma máquina k. A máquina que gerar o menor desbalanceamento é escolhida para processar a tarefa.

Na Figura 4 é ilustrado uma das etapas do algoritmo. Onde, a coluna esquerda de cada máquina define a carga total de processamento e a coluna direita define a carga de comunicação incorrido na máquina. Na figura 4b podemos ver que houve aumento na carga de processamento da máquina M_1 mas, isso não ocorreu para o custo de comunicação haja visto que, neste cenário foi assumido que as duas partições para processar a tarefa na máquina M_1 já se encontrava lá, ou seja, processar a tarefa nesta máquina não resulta em custo de comunicação. Por fim, na figura 4c processar a tarefa na máquina M_2 resultará aumento tanto na carga de consulta quanto na carga de comunicação.

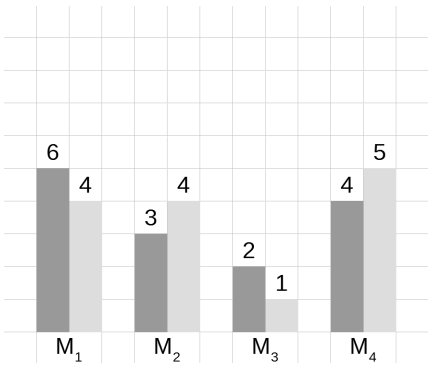
Na Tabela 2 é apresentado o valor do desbalanceamento para cada máquina para a etapa ilustrada na Figura 4. Neste cenário, a máquina M_3 foi a escolhida para processar a tarefa desta etapa pois esta escolha levará a um menor desbalanceamento dos custos envolvidos.

	M_1	M_2	M_3	M_4
cv_{wij}	0,89	0,38	0,29	0,53
cv_{cij}	0,43	0,48	0,20	0,61
$unbalance$	1,32	0,86	0,49	1,14

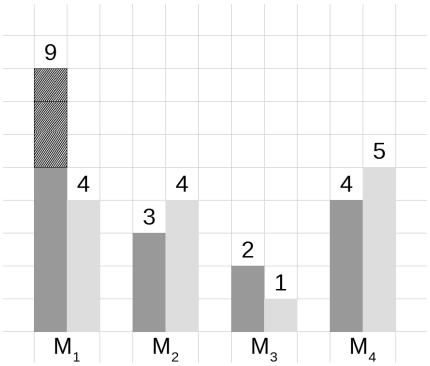
Tabela 2 – Valores do coeficiente de variação da carga de processamento e do custo de comunicação, respectivamente, cv_{wij} e cv_{cij}, e de desbalanceametno, $unbalance$, para cada máquina. Onde, a máquina que gera o menor desbalanceamento é escolhida.

Portanto, a primeira estratégia tem como escolha gulosa selecionar a máquina para processar uma dada tarefa de forma que esta escolha resulte no menor desbalanceamento possível, ou, em outras palavras, distribuir a carga das consultas o mais igual possível mas também balancear o custo de comunicação incorrido ao selecionar estas máquinas.

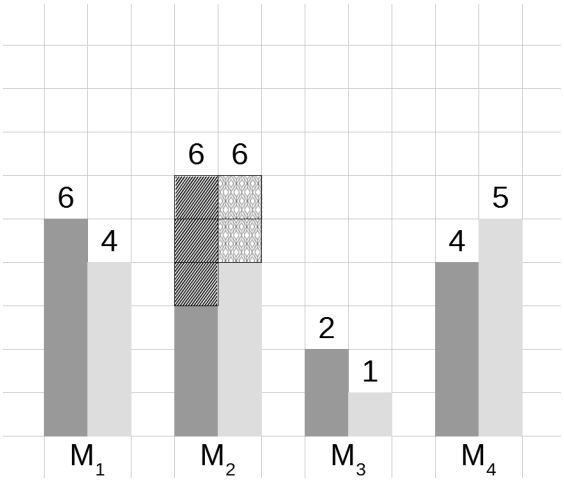
A estratégia anterior proporciona a minimização do *makespan* e o balanceamento do custo de comunicação. Porém, o objetivo é minimizar ambos. Com isso, surge a segunda estratégia, onde ainda utiliza o coeficiente de variação da carga de processamento para



(a) Solução anterior encontrada



(b) Solução caso a tarefa seja escalonada para M_1



(c) Solução caso a tarefa seja escalonada para M_2

Figura 4 – Ilustração da estratégia gulosa para o modelo FM. A coluna de cor cinza escuro representa a carga total de processamento e a coluna de cor cinza claro representa a carga de comunicação incorrida.

calcular o desbalanceamento dos custos envolvidos. Todavia, o cálculo do coeficiente de variação do custo de comunicação será calculado de forma diferente e, somente se, ao selecionar uma máquina, que houver um incremento no *makespan*. Caso não haja incremento no *makespan* usamos o custo de comunicação incorrido normalizado, ou seja, a razão entre o custo de comunicação incorrido e o maior custo de comunicação possível para a tarefa.

O coeficiente de variação da comunicação não será baseado somente no valor da comunicação, mais também no valor do custo de processar a tarefa. Desse modo, o coeficiente de variação será da soma do custo de comunicação com o custo de processar a tarefa. Uma vez que este coeficiente só será utilizado quando houver incremento no *makespan*, ele será alto, diminuindo a chance da máquina ser escolhida para processar a tarefa. Na situação em que processar a tarefa em qualquer máquina gere o aumento do *makespan*, teremos o balanceamento do custo de comunicação no *cluster*.

Assim sendo, a segunda estratégia tem a mesma escolha gulosa do primeira estratégia, com a diferença de buscar colocar a tarefa para processar onde é incorrido o menor custo de comunicação, que corresponde a partição que se encontra na máquina selecionada. Para estas duas estratégias foram propostos dois algoritmos que as implementam.

Além disso, antes de iniciar as etapas do algoritmo, em ambas estratégias, as tarefas são ordenadas em ordem decrescente, onde para cada tarefa é atribuído um valor conforme a seguinte equação:

$$\gamma_j = f * w_j + max_i^m c_{ij} \tag{4.1}$$

Onde, f é um parâmetro que indica a ênfase desejada do quão equilibrado deve ser o escalonamento, w_j é o custo de processar a tarefa j e c_{ij} é o custo de comunicação incorrido para processar a tarefa j na máquina i.

4.3 Escalonamento baseado no coeficiente de variação da distribuição

O Algoritmo 1 implementa a primeira estratégia discutida na Seção 4.2, chamado de escalonamento baseado no coeficiente de variação da distribuição ou pela sigla EBCVD. De modo geral, o algoritmo itera sobre o vetor de tarefas ordenadas pelo gama e calcula o coeficiente de variação da carga de processamento no *cluster* caso a tarefa seja escalonada para a máquina k, assim como o de comunicação, e em seguida os soma. Essa soma nos dá uma ideia do quanto está desbalanceada a distribuição das tarefas pelo *cluster*, nesse caso será escolhida a máquina k, para processar a tarefa, que gere o menor desbalanceamento.

O loop da linha 7 é utilizado para iterar sobre o vetor *gamma_vector* que contém as tarefas ordenadas em ordem decrescente do seu valor gamma calculado pela equação 4.1. Os itens do vetor, *gamma_vector*, nos dá a informação de qual tarefa estamos interessados escalonar. As variáveis $lowest\_unbalance_k$ e *lowest_unbalance* representam, respectivamente, o índice da máquina, cuja a tarefa sendo escalonada para ela gera o menor desbalanceamento, e o valor do desbalanceamento neste cenário. A variável $total\_ca\_cb_{ij}$ representa o valor do custo de comunicação resultante da escolha e w_{ij} é o valor do custo para processar a tarefa. O vetor *pairs* é uma variável global cujo itens são uma estrutura de dados que possuem a informação dos índices das partições envolvidas no processamento da tarefa ij.

O segundo *loop*, na linha 15, tem como função calcular o desbalanceamento para a máquina k caso a tarefa seja escalonada para ela. A variável c_{ij} guarda a informação do custo de comunicação para a tarefa ij ser processada na máquina k. As duas condicionais seguintes estabelecem o fato de que uma partição que se encontra na máquina em que

Algoritmo 1 Escalonamento baseado no coeficiente de variação da distribuição

function EBCVD($n, m, ca, cb, schedule$)

\\ *gamma_vector* é um vetor com as tarefas ordenadas conforme Equação 4.1
\\ *machine_load* e *communication*, são vetores inicializados com zero.
\\ *localyaik* e *localybik* são matrizes inicializadas com zero.

for $i \leftarrow 1, n$ **do**
$ij \leftarrow gamma\_vector[i].ij$
$lowest\_unbalance_k \leftarrow -1$
$lowest\_unbalance \leftarrow \infty$
$total\_ca\_cb_{ij} \leftarrow 0.0$
$w_{ij} \leftarrow f * wij[ij]$
$job \leftarrow pairs[ij]$

for $k \leftarrow 1, m$ **do**
$c_{ij} \leftarrow 0.0$

if $localyaik[k][job.i] = 0$ **then**
$c_{ij} \leftarrow c_{ij} + ca[k][job.i]$
end if

if $localybik[k][job.j] = 0$ **then**
$c_{ij} \leftarrow c_{ij} + cb[k][job.j]$
end if

$cv_{wij} \leftarrow$ CALC_COEFFICIENT_VARIATION$(m, machine_load, k, w_{ij})$
$cv_{cij} \leftarrow$ CALC_COEFFICIENT_VARIATION$(m, communication, k, c_{ij})$
$unbalance \leftarrow cv_{wij} + cv_{cij}$

if $unbalance < lowest\_unbalance$ **then**
$lowest\_unbalance_k \leftarrow k$
$lowest\_unbalance \leftarrow unbalance$
$total\_ca\_cb_{ij} \leftarrow c_{ij}$
end if
end for

$k \leftarrow lowest\_unbalance_k$
$localyaik[k][job.i] \leftarrow 1$
$localybik[k][job.j] \leftarrow 1$

$schedule[k][ij] \leftarrow 1$
$machine\_load[k] \leftarrow machine\_load[k] + w_{ij}$
$communication[k] \leftarrow communication[k] + total\_ca\_cb_{ij}$
end for
end function

a tarefa for escalonada não gera custo de comunicação. Na linha 18 a primeira condição só soma o custo de comunicação para transferir a célula do *dataset* A caso esta não se encontra na máquina k, igualmente acontece na linha 22, porém para a partição do *dataset* B. Logo, as matrizes *localyaik* e *localybik* indicam se uma célula do *dataset* A ou B, respectivamente, já se encontra na máquina, onde as linhas indicam a máquina e as colunas a célula.

Nas linhas 26 e 27 são calculados os coeficientes de variação para as cargas de processamento e de comunicação no *cluster*. Na linha 28 estes coeficientes são somados e o resultado mostra o desbalanceado que está no *cluster* em termo das cargas envolvidas no processamento das tarefas. Na linha 30 verificamos se o desbalanceamento causado ao selecionar a tarefa para ser processada na máquina k é o menor até então, em caso afirmativo, esta será a máquina escolhida. A iteração termina ao fazer estas etapas em todas a máquinas.

Nas linhas 38 e 39 atualizamos as matrizes *localyaik* e *localybik* que significa que as partições envolvidas naquela tarefa foram transferidas para a máquina k, logo, na próxima vez estas partições não incorrerão em custos adicionais de comunicação. Na linha 41 é feita uma atualização na matriz *schedule*, nela, os índices das linhas representam as máquinas k e os índices das colunas representam as tarefas ij, e esta atualização diz que a tarefa ij foi escalonada para a máquina k, setando o valor para 1. Nas linhas 42 e 43 são atualizadas as cargas de processamento e de comunicação para a máquina escolhida. Ao término da execução do algoritmo, a matriz *schedule* estará preenchida com a informação dos escalonamentos das tarefas.

Em sua essência o algoritmo busca escalonar as tarefas de forma que o coeficiente de variação fique o mais próximo possível de zero, o que significa que a distribuição das tarefas no *cluster* está mais homogêneas. Devido a esta propriedade do coeficiente de variação que foi escolhida para a tomada de decisão. Com intuito de tentar melhorar ainda mais os resultados o Algoritmo 2 foi criado com algumas alterações em relação ao Algoritmo 1.

4.4 Escalonamento baseado no coeficiente de variação com comunicação miníma

O Algoritmo 2, referenciado como EBCCM, implementa a segunda estratégia discutida na Seção 4.2. A diferença encontrada neste algoritmo em relação ao anterior é que antes de começar a calcular o desbalanceamento, é encontrado o maior custo de comunicação possível, dentro do laço de repetição iniciado na linha 14, para a tarefa. Este custo será utilizado para normalizar o custo de comunicação, que é calculado pela razão entre o custo de processar a tarefa na máquina k e o maior custo de comunicação possível

para a tarefa.

No *loop* que itera sobre às máquinas para decidir em qual delas a tarefa será processada, foram feitas as seguintes modificações: na linha 46 é calculado o quanto o *makespan* será incrementado, onde um valor maior que zero significa que haverá incremento, do contrário não haverá. O desbalanceamento é então iniciado com o coeficiente de variação da carga de processamento. Se houver um incremento no *makespan*, então o desbalanceamento será somando ao coeficiente de variação da soma do custo de processamento e de comunicação. Do contrário, o desbalanceamento será somado ao custo de comunicação normalizado.

Da linha 64 a 69 são feitas atualizações nas variáveis de controle, já explicado no Algoritmo 1, o importante a ser notado é que agora não temos um vetor com a carga de comunicação para cada máquina, mas, sim um vetor que descreve a carga para cada máquina da soma do custo de processamento e de comunicação denominado de cv_z.

Algoritmo 2 Escalonamento baseado no coeficiente de variação com comunicação miníma

```
1: function EBCCM(n, m, ca, cb, schedule)
2:
3:     gamma_vector ← CALC_GAMMA_IJ_ARRAY(n, m, ca, cb)
4:
5:     for i ← 1, n do
6:         ij ← gama_vector[i].ij
7:         lowest_unbalance_k ← −1
8:         lowest_unbalance ← ∞
9:         total_ca_cb_ij ← 0.0
10:        w_ij ← f ∗ wij[ij]
11:        largest_ca_cb_sum ← −∞
12:        job ← pairs[ij]
13:
14:        for k ← 1, m do
15:            c_ij ← 0.0
16:
17:            if localyaik[k][job.i] = 0 then
18:                c_ij ← c_ij + ca[k][job.i]
19:            end if
20:
21:            if localybik[k][job.j] = 0 then
22:                c_ij ← c_ij + cb[k][job.j]
23:            end if
24:
25:            if c_ij > largest_ca_cb_sum then
26:                largest_ca_cb_sum ← c_ij
27:            end if
28:        end for
29:
30:        if largest_ca_cb_sum = 0.0 then
31:            largest_ca_cb_sum ← 1
32:        end if
33:
```

Algoritmo 3 EBCCM: Continuação

```
34:         for k ← 1, m do
35:             c_ij ← 0.0
36:
37:             if localyaik[k][job.i] = 0 then
38:                 c_ij ← c_ij + ca[k][job.i]
39:             end if
40:
41:             if localybik[k][job.j] = 0 then
42:                 c_ij ← c_ij + cb[k][job.j]
43:             end if
44:
45:             wij_cv ← CALC_COEFFICIENT_VARIATION(m, machine_load, k, w_ij)
46:             mkspaninc ← machine_load[k] + w_ij − makespan
47:             unbalance ← wij_cv
48:
49:             if mkspaninc > 0 then
50:                 cw_ij ← c_ij + w_ij
51:                 unbalance ← unbalance+ CALC_COEFFICIENT_VARIATION(m, cv_z, k, cw_ij)
52:             else
53:                 unbalance ← unbalance + c_ij/largest_ca_cb_sum
54:             end if
55:
56:             if unbalance < lowest_unbalance then
57:                 lowest_unbalance_k ← k
58:                 lowest_unbalance ← unbalance
59:                 total_ca_cb_ij ← c_ij
60:             end if
61:         end for
62:
63:         k ← lowest_unbalance_k
64:         localyaik[k][job.i] ← 1
65:         localybik[k][job.j] ← 1
66:         schedule[k][ij] ← 1
67:
68:         machine_load[k] ← machine_load[k] + w_ij
69:         cv_z[k] ← cv_z[k] + total_ca_cb_ij + w_ij
70:
71:         if machine_load[k] > makespan then
72:             makespan ← machine_load[k]
73:         end if
74:     end for
75: end function
```

4.5 Cálculo do coeficiente de variação

Os Algoritmos 1 e 2 utilizaram o Algoritmo 4 para calcular o novo coeficiente de variação, caso a máquina k seja selecionada para processar uma tarefa. Onde, m é quantidade de máquinas, que compõe o *cluster*, *data* é um vetor com os valores de interesse atualizados, por exemplo, a carga de processamento de cada máquina, e $increment_k$ é o número da máquina que talvez será escolhida e *increment* é o valor que será incrementado caso a máquina seja escolhida, por exemplo, se a máquina for escolhida, então, a sua carga de processamento será incrementado.

Algoritmo 4 Cálculo do Coeficiente de Variação

```
1:  function CALC_COEFFICIENT_VARIATION(m, data, increment_k, increment)
2:
3:      sum ← 0.0
4:      mean ← 0.0
5:      variance ← 0.0
6:      data_incremented ← data[increment_k] + increment
7:
8:      for k ← 1, m do
9:          if increment_k = k then
10:             sum ← sum + data_incremented
11:         else
12:             sum ← sum + data[k]
13:         end if
14:     end for
15:
16:     mean ← sum/m
17:
18:     for k ← 1, m do
19:         if increment_k = k then
20:             variance ← variance + pow(data_incremented − mean, 2)
21:         else
22:             variance ← variance + pow(data[k] − mean, 2)
23:         end if
24:     end for
25:
26:     standard_deviation ← sqrt(variance/m)
27:     coefficient_variation ← standard_deviation/mean
28:
29:     return coefficient_variation
30: end function
```

5 AVALIAÇÃO E RESULTADOS

5.1 Introdução

Neste capítulo são apresentados a avaliação dos algoritmos desenvolvidos e os resultados obtidos.

5.2 Procedimentos Metodológicos

Nesta seção, serão descritos os dados e as consultas de junção espacial que foram utilizadas para realizar a experimentação e também como foi conduzida a análise dos dados provenientes dos experimentos realizados.

5.2.1 Materiais utilizados

Com intuito de conduzir a avaliação, os *datasets* especificados nas Tabelas 3 e 4 foram escolhidos para serem utilizados no experimento. Para avaliar os algoritmos desenvolvidos neste trabalho foram utilizadas as consultas de junção espacial descritas na Tabela 5. Esta tabela mostra os detalhes das consultas de C_1 a C_{20}, os *datasets* envolvidos e quantidade de tarefas necessárias para computar a consulta.

Por exemplo, a consulta C_1 envolve os *datasets*, descritos na tabela 3, A e H e resulta em 8.082 tarefas para processá-la. Além disso, nas duas últimas colunas da tabela são apresentados os tamanhos dos dois *datasets* envolvidos na consulta.

Tabela 3 – Datasets brasileiros - IBGE e LAPIG

Nome	*Abrev.*	*Tipo*	*Cardinalidade*	*Arquivo SHP Tamanho (MB)*
Alertas de incêndio	A	Polígonos	32.578	11,2
Hidrografia	H	Linhas	226.963	64,5
Estradas	R	Linhas	51.646	15,2
Municípios	C	Polígonos	5.564	38,8
Vegetação	V	Polígonos	2.140	4,7

O escalonamento será realizado para m máquinas tal que $m \in \{4, 8, 16, 32\}$ resultando em um total de 80 escalonamento para os algoritmos que foram desenvolvidos neste trabalho. A máquina na qual os experimentos foram executados possui uma configuração com 4 GB de RAM e um processador Intel Core i5-3330 CPU 3.00GHz x 4. Além disso,

Tabela 4 – Datasets mundiais - DCW

Nome	*Abrev.*	*Tipo*	*Cardinalidade*	*Arquivo SHP Tamanho (MB)*
Rios	RI	Linhas	943.638	243,2
Trilhos	RA	Linhas	194.261	28,7
Hidrografia Interior	HI	Polígonos	338.860	136,7
Contorno de elevação	EC	Linhas	703.574	572,5
Cultivo	CR	Polígonos	123.746	69,3

Tabela 5 – Junções espaciais escalonadas no experimento.

Nome	*Consulta*	*N. Tarefas (Jobs)*	Tam. do dataset A	Tam. do dataset B
C_1	$A \bowtie H$	14.754	8.082	1.696
C_2	$A \bowtie R$	11.644	8.082	620
C_3	$A \bowtie C$	8.702	8.082	33
C_4	$A \bowtie V$	8.444	8.082	15
C_5	$H \bowtie R$	16.582	7.125	2.293
C_6	$H \bowtie C$	9.209	7.587	103
C_7	$H \bowtie V$	8.631	7.755	36
C_8	$R \bowtie C$	2.968	2.139	105
C_9	$R \bowtie V$	2.578	2.160	36
C_{10}	$C \bowtie V$	244	114	37
C_{11}	$RI \bowtie RA$	17.122	5.572	4.632
C_{12}	$RI \bowtie HI$	35.182	10.285	8.808
C_{13}	$RI \bowtie EC$	33.340	10.019	8.056
C_{14}	$RI \bowtie CR$	18.860	6.630	4.115
C_{15}	$RA \bowtie HI$	15.666	4.612	4.796
C_{16}	$RA \bowtie EC$	15.140	4.588	4.422
C_{17}	$RA \bowtie CR$	12.173	4.209	3.211
C_{18}	$HI \bowtie EC$	29.750	8.477	7.940
C_{19}	$HI \bowtie CR$	16.634	5.577	4.060
C_{20}	$EC \bowtie CR$	15.886	5.106	3.934

foi obtido com Oliveira (2017) o código fonte dos métodos LR e LP e nesse código foram implementados os algoritmos EBCVD e EBCVDMC.

5.2.2 Avaliação

Na tese de Oliveira (2017) foi calculado o limite inferior Z^{lb}_{FM} para o valor ótimo Z^{*}_{FM} com intuito de verificar a performance dos escalonamentos gerados pelo modelo SM em comparação com o modelo FM. O cálculo da distância (*gap*) entre a solução ótima para o modelo SM e o FM é realizada pela formula $(Z^{+}_{FM} - Z^{lb}_{FM})/Z^{lb}_{FM}$, onde o sinal $+$ indica o método usado para escalonar.

Uma vez que tenha os resultados dos métodos citados e o valor para Z^{lb}_{FM} neste

projeto, após realizados os escalonamentos, foram calculadas as distâncias para os algoritmos que foram propostos nesta monografia. A partir desses resultados foram feitas análises para verificarmos como os algoritmos se saíram comparados aos existentes.

5.3 Resultado do experimento

Nesta seção fazemos uma comparação entre as distâncias dos resultados obtidos pelos algoritmos desenvolvidos neste trabalho e os algoritmos propostos em Oliveira (2017), na Figura 5 é mostrado para cada algoritmo o *gap* que há entre a solução encontrada pelo método em relação ao limite inferior para o modelo FM. Para melhorar a visualização, limitamos a escala do *gap* em 100% e algumas instâncias não foram mostradas adequadamente (Ex. C_{11}). Uma listagem completa dos valores obtidos e que foram usados para gerar o gráfico está disponível na Tabela A, Apêndice A.

Na Figura 5 há quatro gráficos, cada um para um número de máquinas m, onde $m \in \{4, 8, 16, 32\}$. Há quatro símbolos, cada um indicando um método e seu *gap* para cada consulta. Em geral, o método que apresentou o menor *gap* foi o LR. O método EBCVD foi, em geral, o que apresentou o maior *gap*. Percebe-se ainda que, em geral, a medida em que m aumenta, o *gap* do método LP também aumenta. A Tabela 6 evidencia melhor este comportamento que também foi observado no trabalho de (OLIVEIRA, 2017) e que ocorre devido ao maior número de tarefas escalonadas de forma fracionada pela relaxação linear.

Dos dois métodos propostos neste trabalho, o melhor foi o EBCVDMC. O interessante a ser notado é que, em geral, a medida em que m aumenta o seu *gap* diminui, ao contrário do LP, como é mostrado na Tabela 6, o que vale ressaltar ainda, que o método EBCVDMC se saiu melhor em 68 instâncias comparado ao método LP. A média variou de $15,75\%$, com $m = 4$, para $7,22\%$, com $m = 32$. Neste cenário, o método EBCVDMC se sai melhor que o método LP a medida em que m aumenta. Em comparação ao LR, o EBCVDMC se saiu melhor em quatro instâncias, sendo elas a consulta C_{11} para $m = 4$, C_{11} para $m = 8$, C_1 e C_2 para $m = 32$.

Podemos notar ainda que a medida em que m aumenta os resultados do metodo EBCVDMC se aproximam dos resultados do método LR. Para investigar este comportamento foi calculado a média e o desvio padrão das diferenças entre os *gaps* dos dois métodos para cada $m \in \{4, 8, 16, 32\}$. O resultado, apresentado na Tabela 7, evidencia esta redução para algumas instâncias. A média da diferença entre os dois métodos varia de $10,72\%$, com $m = 4$, para $5,00\%$, com $m = 32$. Porém, o desvio padrão alto indica que a redução não é a mesma para todas as instâncias.

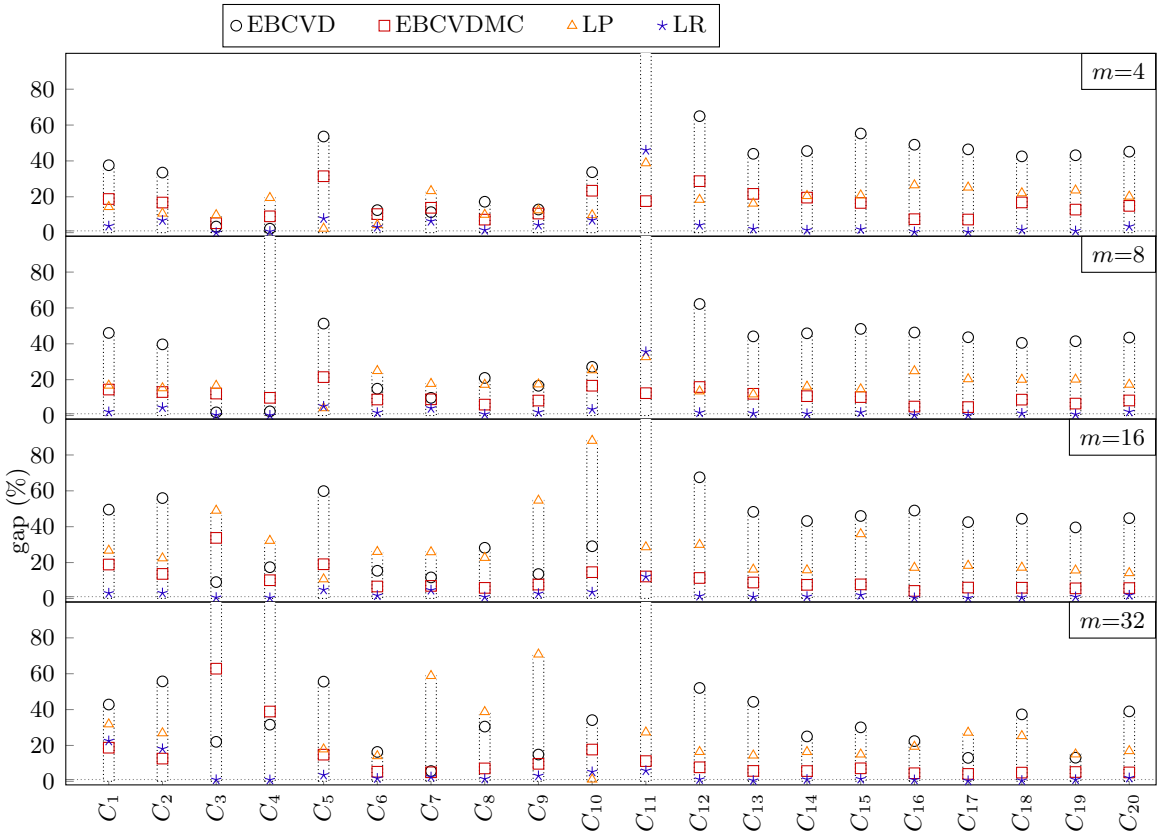


Figura 5 – *Gap* entre cada escalonamento realizado por EBCVD, EBCVDMC, LP, e LR.

Tabela 6 – Média e desvio padrão das distâncias dos resultados em relação ao limite inferior para o modelo FM.

Método	Média para m				Desvio padrão para m			
	4	8	16	32	4	8	16	32
LR	2,56	1,60	1,44	1,30	9,93	7,68	6,21	1,91
LP	18,90	17,36	24,28	22,33	19,50	8,75	37,99	16,88
EBCVD	42,85	42,47	43,81	31,06	31,95	28,60	30,36	32,44
EBCVDMC	15,75	10,08	7,78	7,22	6,08	6,79	14,02	3,92

Tabela 7 – Média e desvio padrão da diferença entre os *gaps* dos métodos EBCVDMC e LR.

Média para m				Desvio padrão para m			
4	8	16	32	4	8	16	32
10,72	8,13	5,90	5,00	10,85	7,93	7,05	15,08

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, estudou-se os modelos FM e SM para escalonamento de tarefas de multijunção espacial e propôs-se um novo algoritmo guloso, chamado EBCVDMC, capaz de encontrar soluções viáveis para o modelo FM.

O algoritmo proposto possui complexidade polinomial $O(nm^2)$, sendo n a quantidade tarefas e m o número de máquinas, e foi comparado com outros métodos baseados em técnicas de otimização combinatória, como Relaxação Linear (LP) e Lagrangiana (LR).

Comparado ao método LP, o algoritmo proposto produziu melhores soluções para 68 das 80 instâncias avaliadas. Comparado ao método LR, o algoritmo apresentou soluções competitivas em relação ao valor ótimo, sendo 4 delas melhores. O LR, mesmo sendo aplicado ao SM, se mostrou superior, apesar de ter uma complexidade maior.

De forma geral, devido a sua complexidade, o método EBCVDMC é um candidato para escalonar instâncias de consultas com baixo tempo de execução, onde o tempo de escalonamento também precisa ser pequeno para ser vantajoso.

Como trabalho futuro, é interessante explorar técnicas mais sofisticadas de construção de algoritmos gulosos. Uma delas poderia ser a utilização de grafos. Com um grafo pode-se representar a característica das tarefas e o compartilhamento de dados, e com isso, explorar algoritmos de pontes e cortes em grafos para reduzir o custo de comunicação e *makespan*.

REFERÊNCIAS

AJI, A.; WANG, F.; SALTZ, J. H. Towards building a high performance spatial query system for large scale medical imaging data. In: *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. New York, NY, USA: ACM, 2012. (SIGSPATIAL '12), p. 309–318. ISBN 978-1-4503-1691-0. Disponível em: <http://doi.acm.org/10.1145/2424321.2424361>. Citado na página 16.

ANGEL, E.; BAMPIS, E.; KONONOV, A. A fptas for approximating the unrelated parallel machines scheduling problem with costs. In: SPRINGER. *European Symposium on Algorithms*. [S.l.], 2001. p. 194–205. Citado na página 30.

CAMPBELL, M. S. J. E. *Geographic information system basics*. first. [S.l.]: Creative Commons, v. 1.0, 2012. Citado 5 vezes nas páginas 9, 20, 21, 22 e 23.

CHENG, B. et al. Hypergraph+: An improved hypergraph-based task-scheduling algorithm for massive spatial data processing on master-slave platforms. *ISPRS International Journal of Geo-Information*, Multidisciplinary Digital Publishing Institute, v. 5, n. 8, p. 141, 2016. Citado na página 30.

CUNHA, A. R. et al. Distributed spatial join processing for multiple spatial datasets - multi-way spatial join. In: *XXXIII Brazilian Symposium on Computer Networks and Distributed Systems, SBRC 2015, Vitória, Brazil, May 18-22, 2015*. IEEE Computer Society, 2015. p. 171–181. Disponível em: <https://doi.org/10.1109/SBRC.2015.29>. Citado na página 16.

FITZ, P. R. *Geoprocessamento sem complicação*. [S.l.]: Oficina de textos, 2008. ISBN 978-8586238826. Citado 3 vezes nas páginas 20, 21 e 22.

FORNARI, M.; COMBA, J. L. D.; IOCHPE, C. A rule-based optimizer for spatial join algorithms. In: *Advances in Geoinformatics*. [S.l.]: Springer, 2007. p. 87–106. Citado na página 25.

HUISMAN, O.; BY, R. D. Principles of geographic information systems. *ITC Educational Textbook Series*, v. 1, p. 17, 2009. Citado na página 22.

JACOX, E. H.; SAMET, H. Spatial join techniques. *ACM Transactions on Database Systems (TODS)*, Acm, v. 32, n. 1, p. 7, 2007. Citado na página 23.

LEVITIN, A. V. Introduction to the design and analysis of algorithms. In: _____. 3. ed. [S.l.]: Pearson, 2012. cap. 9 - Greedy Technique. ISBN 9780132316811. Citado na página 28.

MAMOULIS, N.; PAPADIAS, D. Selectivity estimation of complex spatial queries. In: JENSEN, C. S. et al. (Ed.). *Advances in Spatial and Temporal Databases, 7th International Symposium, SSTD 2001, Redondo Beach, CA, USA, July 12-15, 2001, Proceedings*. Springer, 2001. (Lecture Notes in Computer Science, v. 2121), p. 155–174. Disponível em: <https://doi.org/10.1007/3-540-47724-1_9>. Citado na página 16.

OLIVEIRA, S. de et al. Processamento distribuído de operações de junção espacial com bases de dados dinâmicas para análise de informações geográficas. *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2013. Citado na página 16.

OLIVEIRA, T. B. d. Efficient processing of multiway spatial join queries in distributed systems. Universidade Federal de Goiás, 2017. Citado 10 vezes nas páginas 16, 17, 24, 25, 26, 27, 28, 31, 44 e 45.

OLIVEIRA, T. B. de; COSTA, F. M.; RODRIGUES, V. J. do S. Definição de planos de execução distribuídos para consultas de junção espacial usando histogramas multidimensionais. In: BRAGANHOLO, V. (Ed.). *XXX Simpósio Brasileiro de Banco de Dados, SBBD 2015, Petrópolis, Rio de Janeiro, Brasil, October 13-16, 2015*. SBC, 2015. p. 89–100. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/sbbd/2015/007.pdf>. Citado 2 vezes nas páginas 16 e 17.

PRASAD, S. K. et al. Parallel processing over spatial-temporal datasets from geo, bio, climate and social science communities: A research roadmap. In: IEEE. *2017 IEEE International Congress on Big Data (BigData Congress)*. [S.l.], 2017. p. 232–250. Citado na página 25.

PURI, S.; PAUDEL, A.; PRASAD, S. K. Mpi-vector-io: parallel i/o and partitioning for geospatial vector data. In: *Proceedings of the 47th International Conference on Parallel Processing, ICPP*. [S.l.: s.n.], 2018. p. 13. Citado 2 vezes nas páginas 24 e 26.

RIGAUX, P.; SCHOLL, M.; VOISARD, A. *Spatial databases: with application to GIS*. [S.l.]: Morgan Kaufmann Publishers, 2002. ISBN 1558605886. Citado 2 vezes nas páginas 21 e 23.

SHCHEPIN, E. V.; VAKHANIA, N. An optimal rounding gives a better approximation for scheduling unrelated machines. *Operations Research Letters*, Elsevier, v. 33, n. 2, p. 127–133, 2005. Citado na página 29.

VAZIRANI, V. V. *Approximation Algorithms*. [S.l.]: Springer Science & Business Media, 2001. ISBN 3-540-65367-8. Citado na página 29.

YU, J.; WU, J.; SARWAT, M. Geospark: A cluster computing framework for processing large-scale spatial data. In: ACM. *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. [S.l.], 2015. p. 70. Citado na página 25.

APÊNDICE A – TABELA COM OS RESULTADOS DOS *GAPS*

Tabela 8 – Resultado dos *gap* para cada consulta e valor de $m \in \{4, 8, 16, 32\}$. A última coluna indica a diferença entre o *gap* do EBCVDMC e LR, onde valores negativos indicam que o metodo EBCVDMC foi melhor o que o metodo LR.

Consulta	m	LR	LP	EBCVD	EBCVDMC	EBCVDMC - LR
1	4	3,66	14,37	37,60	18,76	15,10
2	4	6,82	10,73	33,50	16,78	9,97
3	4	0,02	9,84	3,39	5,25	5,23
4	4	0,62	19,41	2,18	9,10	8,48
5	4	7,91	2,30	53,62	31,46	23,55
6	4	3,06	4,32	12,55	10,34	7,28
7	4	6,37	23,26	11,43	13,86	7,48
8	4	1,37	10,04	17,23	7,26	5,89
9	4	4,18	12,53	12,90	10,65	6,47
10	4	6,92	9,83	33,70	23,38	16,46
11	4	45,97	38,78	152,42	17,58	-28,39
12	4	4,14	18,40	64,96	28,71	24,58
13	4	2,05	16,28	43,96	21,62	19,56
14	4	1,37	20,51	45,53	19,53	18,16
15	4	1,79	20,86	55,28	16,57	14,78
16	4	0,46	26,50	49,03	7,51	7,05
17	4	0,33	25,23	46,44	7,31	6,98
18	4	1,51	22,05	42,54	16,85	15,34
19	4	0,94	23,61	43,16	12,85	11,91
20	4	3,45	20,16	45,16	14,93	11,48
1	8	2,03	16,75	46,06	14,49	12,46
2	8	4,46	15,30	39,69	13,11	8,65
3	8	0,25	16,67	1,77	12,26	12,01
4	8	0,09	100,67	2,33	9,88	9,79
5	8	5,16	4,18	51,28	21,43	16,27
6	8	1,58	24,91	14,96	8,94	7,36
7	8	4,04	17,73	9,87	9,05	5,01
8	8	0,70	17,23	20,98	6,09	5,39
9	8	1,81	17,45	16,59	8,35	6,54

Consulta	m	LR	LP	EBCVD	EBCVDMC	EBCVD - EBCVDMC
10	8	3,43	25,33	27,14	16,63	13,20
11	8	35,58	32,66	142,56	12,47	-23,11
12	8	1,62	13,55	62,21	15,90	14,28
13	8	1,20	11,87	44,15	12,12	10,92
14	8	0,95	16,18	45,86	10,80	9,85
15	8	1,68	14,72	48,32	10,29	8,61
16	8	0,33	24,82	46,31	5,00	4,67
17	8	0,24	20,34	43,66	4,70	4,46
18	8	1,21	20,05	40,51	8,86	7,66
19	8	0,61	20,14	41,45	6,66	6,05
20	8	2,05	17,28	43,48	8,40	6,35
1	16	2,91	26,73	49,53	18,90	15,99
2	16	3,01	22,45	55,96	13,66	10,65
3	16	0,47	49,01	9,10	33,76	33,28
4	16	0,33	32,20	17,41	10,15	9,82
5	16	4,81	10,75	59,85	19,06	14,25
6	16	1,58	26,01	15,41	6,65	5,07
7	16	4,62	25,86	11,72	6,99	2,37
8	16	0,77	22,69	28,23	5,88	5,11
9	16	2,66	54,64	13,59	7,72	5,06
10	16	3,50	87,97	29,09	14,64	11,14
11	16	12,11	28,63	155,48	12,24	0,13
12	16	1,30	29,97	67,55	11,40	10,10
13	16	0,79	16,16	48,28	8,87	8,08
14	16	0,98	15,79	43,19	7,58	6,60
15	16	1,83	35,92	45,98	7,83	6,00
16	16	0,32	17,18	49,02	4,18	3,86
17	16	0,24	18,28	42,55	6,05	5,81
18	16	0,58	17,23	44,42	5,96	5,38
19	16	0,85	15,68	39,60	5,62	4,77
20	16	1,77	14,20	44,75	5,76	3,99
1	32	22,53	31,86	42,82	18,75	-3,79
2	32	18,14	26,88	55,73	12,67	-5,47
3	32	0,87	120,23	22,02	62,81	61,94
4	32	0,78	152,02	31,59	38,92	38,15
5	32	3,48	17,89	55,60	14,87	11,38
6	32	1,62	14,12	16,38	5,33	3,71
7	32	2,40	58,82	5,72	5,04	2,64

Consulta	m	LR	LP	EBCVD	EBCVDMC	EBCVD - EBCVDMC
8	32	1,32	38,70	30,52	7,19	5,88
9	32	3,08	70,81	14,92	9,75	6,67
10	32	5,28	1,07	34,12	17,71	12,42
11	32	5,93	27,33	133,29	11,27	5,33
12	32	1,06	16,45	52,07	7,82	6,76
13	32	0,44	14,47	44,32	5,79	5,35
14	32	1,04	16,42	25,03	5,71	4,67
15	32	1,28	15,04	30,09	7,25	5,97
16	32	0,78	19,37	22,43	4,50	3,72
17	32	0,30	27,20	13,07	4,12	3,82
18	32	0,46	25,29	37,33	4,85	4,39
19	32	0,92	15,09	13,17	5,18	4,26
20	32	1,88	16,93	39,00	4,95	3,06