FRANK WILLIAN RODRIGUES DA SILVA

IMPACTO DO USO DE DATA NODES E WORKER NODES NO PROCESSAMENTO PARALELO DE CONSULTAS SOBRE GRANDES VOLUMES DE DADOS

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre.
Área de Concentração: ENGENHARIA DE SISTEMAS E INFORMAÇÃO.

Orientadora: Profª. Dra. Vanessa Braganholo Murta
Coorientador: Prof. Dr. Victor Teixeira de Almeida

Niterói
2018

Ficha catalográfica automática - SDC/BEE

S586i Silva, Frank Willian Rodrigues da
IMPACTO DO USO DE DATA NODES E WORKER NODES NO PROCESSAMENTO PARALELO DE CONSULTAS SOBRE GRANDES VOLUMES DE DADOS / Frank Willian Rodrigues da Silva ; Vanessa Braganholo, orientadora ; Victor Teixeira de Almeida, coorientador. Niterói, 2018.
126 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.m.03037392150

1. Processamento Paralelo. 2. Data Node e Worker Node. 3. Big Data. 4. Banco de Dados. 5. Produção intelectual. I. Título II. Braganholo,Vanessa, orientadora. III. Almeida, Victor Teixeira de, coorientador. IV. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

FRANK WILLIAN RODRIGUES DA SILVA

IMPACTO DO USO DE DATA NODES E WORKER NODES NO PROCESSAMENTO PARALELO DE CONSULTAS SOBRE GRANDES VOLUMES DE DADOS

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre.
Área de Concentração: ENGENHARIA DE SISTEMAS E INFORMAÇÃO.

Aprovada em Novembro de 2018.

BANCA EXAMINADORA

Prof. Dr. Vanessa Braganholo Murta – Orientador
Universidade Federal Fluminense

Prof. Dr. Victor Teixeira de Almeida – Coorientador
Universidade Federal Fluminense

Prof. Dr. José Antônio Fernandes de Macêdo
Universidade Federal do Ceará

Prof. Dr. Maria Cristina Silva Boeres
Universidade Federal Fluminense

Niterói
2018

Dedico esta dissertação à Universidade do Estado de Mato Grosso que me proporcionou a oportunidade desta qualificação, bem como à minha família e amigos.

AGRADECIMENTOS

Agradeço primeiramente a Universidade do Estado de Mato Grosso por proporcionar a oportunidade de qualificação de seus funcionários e também por acatar minha solicitação de afastamento, para qualificação nível Mestre, no Programa de Pós-Graduação do Instituto de Computação da Universidade Federal Fluminense. Agradeço aos meus orientadores pela oportunidade de pesquisa, pelos conhecimentos adquiridos a cada reunião, pela disponibilidade e pela atenção com minhas limitações. Aos professores do Instituto, meu imenso agradecimento pela atenção e conhecimentos passados, e aos funcionários meus agradecimentos pela atenção e carinho. Agradeço de todo o coração a minha família que me apoiou na distância para suportar a saudade e me deu forças para enfrentar esta etapa da minha vida. Da mesma forma, agradeço aos amigos antigos e novos, de diversos locais do país, que me ajudaram a suportar a estadia longe de casa e também pela disponibilidade com minhas limitações acadêmicas.

RESUMO

Sistemas de gerência de Big Data, em sua maioria, utilizam *clusters* para processamento massivo paralelo de consultas. As arquiteturas destes sistemas variam. Alguns sistemas gerenciam cada máquina como sendo um único nó com disco de dados acoplado ou acesso a um *storage* compartilhado. Outros sistemas alocam vários nós em uma mesma máquina explorando vários núcleos de processamento com fatiamento do recurso de armazenamento para cada nó, ou limitam o número de nós alocados na mesma máquina ao número de controladoras de disco que a máquina possui. Nos casos em que a máquina possui mais processadores do que controladoras de disco, esses processadores ficam ociosos. Em nossa pesquisa bibliográfica não encontramos nenhuma estratégia que explore totalmente os recursos disponíveis de armazenamento e processamento de forma independente em uma mesma máquina. Dessa forma, as abordagens existentes são incapazes de explorar núcleos ociosos de processadores para atuarem de forma independente no processamento paralelo de operadores da consulta. Neste contexto, esta dissertação investiga o impacto da exploração de arquiteturas *multi-core* no processamento paralelo de consultas. Isso é feito por meio do uso de *worker nodes*, que realizam processamento e não armazenam dados, e *data nodes*, que processam e armazenam dados, alocados em uma mesma máquina utilizando o mecanismo de execução MyriaX, componente do sistema de gerência de Big Data Myria. Executamos e avaliamos dois experimentos (I e II) atacando diferentes pontos cruciais ao desempenho do processamento paralelo de consultas analíticas em *cluster* utilizando diversas configurações e alocações de *data nodes* e *worker nodes* (cenários). Em ambos os experimentos, o cenário *Baseline* se refere à configuração padrão de nós do MyriaX e os cenários *Avaliação* se referem a configuração de nós com adição de *worker nodes* em uma mesma máquina. O Experimento I avaliou o processamento de duas consultas diferentes, uma de Auto-junção (C1) e outra de Identificação de Triângulos (C2), sobre uma base de dados da rede social Twitter com grande volume de dados. A partir dos resultados deste experimento, identificamos que cenários *Avaliação* causaram significativa aceleração no tempo de processamento de ambas as consultas, mas há uma deterioração a partir de um determinado ponto no acréscimo de *worker nodes* para a consulta de C2. O Experimento II avaliou o desempenho destes cenários no processamento de uma carga de trabalho composta por 11 consultas selecionadas do benchmark TPC-DS. Os resultados deste experimento mostram aceleração na maioria dos cenários *Avaliação* das consultas individuais. Mas, assim como no Experimento I e também para os resultados da carga de trabalho, estes resultados apresentaram uma curva entre melhora e deterioração no tempo de processamento. Os resultados destes experimentos mostram que a característica da consulta precisa ser levada em conta na escolha do melhor cenário de execução, e abre oportunidades para elaboração de heurísticas que possam ser usadas na geração do plano da consulta e escolher a quantidade de *worker nodes* que deve ser usada para processar a consulta.

Palavras-chave: Big Data, Banco de Dados, Processamento Paralelo, Data Node, Worker Node.

ABSTRACT

Most of Big Data management systems perform massive parallel query processing on clusters. Their architectures vary. Some systems manage each machine as a single node with local hard disk or shared storage access. Other systems deploy many nodes on the same machine exploring the multi-core approach. On such systems, each node accesses slices of the same storage resource, or the system limits the number of nodes that can be allocated in a machine to the number of disk controllers that machine has. In the cases the machine has more processors than disk controllers, these processors are kept idle (are not allocated as processing nodes). To the best of our knowledge, none of these architectures explore all of the available CPU cores and storage resources independently in the same machine. These systems are then unable to explore CPU cores to process parallel query operators independently. In this dissertation, we investigate the impact of exploring multi-core architectures on query parallel processing. We do this by using worker nodes, which process data but do not store data (that is, they do not have access to a disk), and data nodes, which process and store data, deployed on the same machine using the MyriaX engine of the Myria Big Data Management System. We perform two experiments (I and II) addressing key points in the performance of parallel processing analytical queries on a cluster, using different allocation configurations for worker and data nodes. On both experiments, the Baseline configuration refers to the MyriaX default nodes configuration, and the *Avaliação* configuration refers to the configuration with additional worker nodes on the same machine. Experiment I evaluates the processing of two queries: a self-join (C1) and a triangles counting (C2), on a large Twitter dataset. From the results of this experiment, we identify that *Avaliação* configurations achieved speedup on both queries but caused speed down on query C2 after adding a certain amount of worker nodes. Experiment II evaluates the performance of the Baseline and *Avaliação* configurations to process a workload composed of 11 queries of the TPC-DS benchmark. The results of this experiment show speedup on most of the *Avaliação* configurations of the single queries. However, like on Experiment I and workload results, these results show a curve with speedup and speed down. The results of these experiments show that the characteristics of the queries need to be taken into consideration when choosing the best execution configuration and opens up opportunities to the development of heuristics that can be used to generate the query plan and choose the amount of worker nodes that should be used.

Keywords: Big Data, Database, Parallel Processing, Worker Node, Data Node.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

Uma grande quantidade de dados está sendo gerada diariamente através de redes sociais, blogs, comunidades online, bem como fontes de notícias e aplicativos móveis. Organizações e pesquisadores de vários domínios reconhecem o enorme valor e conhecimento que pode ser adquirido pela captura destes dados, tornando-os disponíveis para consultas e análises. Este é um dos principais focos do movimento Big Data atualmente. A disponibilidade temporal de informações por sistemas de Big Data pode ser altamente benéfica para diversas áreas incluindo segurança e saúde pública, segurança nacional, medicina, marketing, ciência política e elaboração de políticas governamentais, por exemplo (ALSUBAIEE et al., 2012).

A necessidade de análise de Big Data continua a crescer na indústria, governo e ciência. Isto levou empresas e organizações a buscarem alternativas com melhor custo/benefício, pois sistemas tradicionais de banco de dados tornaram-se opções pouco atrativas para o processamento de dados de ambas as origens. Motivadas pelo surgimento de plataformas de nuvem que utilizam *clusters* de centenas ou milhares de máquinas de propósito geral e também pelo aumento do poder de computação com a tecnologia *multi-core*, estas empresas e organizações contribuíram para o surgimento de diversos sistemas de gerência de Big Data.

Nesta dissertação, avaliamos sistemas para processamento de consultas sobre grandes volumes de dados. Estes sistemas, em sua maioria, utilizam *clusters* de máquinas de propósito geral para distribuição dos dados e processamento massivamente paralelo de consultas. As arquiteturas utilizadas por eles variam; alguns exigem que cada máquina tenha um disco acoplado, tais como Apache Hadoop[1], HadoopDB (ABOUZEID et al., 2009), Apache Spark[2] (ZAHARIA et al., 2012), Asterix (ALSUBAIEE et al., 2014), Impala (BITTORF et al., 2015), ElasTras (DAS et al., 2013), Pregel (MALEWICZ et al., 2010), Dryad (ISARD et al., 2007), Vertica[3] (LAMB et al., 2012), Presto[4] (GHEMAWAT et al., 2003), ParGRES (MATTOSO et al., 2005), MyriaX (WANG, Jingjing et al., 2017). Já o SnowFlake (DAGEVILLE et al., 2016) utiliza de uma base de dados compartilhada que todos os nós acessam de forma concorrente. Outros sistemas exploram a tecnologia *multi-core* alocando vários nós em uma mesma máquina, tais como Nephele (WARNEKE; KAO, 2009), Amazon Redshift (GUPTA et al., 2015) e Greenplum[5].

[1] http://hadoop.apache.org/
[2] https://spark.apache.org/
[3] https://my.vertica.com/
[4] http://prestodb.io/
[5] http://greenplum.org

Apesar de existirem diversos sistemas para suporte a análise e gerência de Big Data, ainda há espaço para melhorias em termos de desempenho. Isso ocorre porque a maioria destes sistemas utiliza cada máquina como sendo um único tipo de nó com recurso de processamento e armazenamento. Mesmo que cada máquina possua diversos processadores, cada um com diversos núcleos, e vários discos de dados, eles são gerenciados pelo sistema como um único recurso. Apesar de alguns sistemas explorarem os vários núcleos de processamento disponíveis, há fatiamento do recurso de armazenamento para cada nó, ou seja, há uma partição de dados associada a cada nó de processamento gerando gargalos de I/O.

Atualmente, é comum encontrar máquinas com mais núcleos de processamento que controladoras de disco de dados. Assim, se cada núcleo de processamento for associado a uma controladora de disco para evitar gargalos de I/O (o que chamamos de *data nodes* (nós que processam e armazenam dados)), ainda restariam núcleos ociosos. No entanto, sabe-se que o uso de mais núcleos implicam um melhor potencial de escalabilidade para operadores relacionais, tais como junções (KIM et al., 2009). Esses núcleos de processamento ociosos poderiam atuar no *pipeline* de processamento de consultas como *worker nodes* (nós que processam mas não armazenam dados).

Em nossa pesquisa bibliográfica, não encontramos nenhuma estratégia que explore totalmente os recursos disponíveis de armazenamento e processamento de forma independente em uma mesma máquina, sendo incapazes de explorar núcleos ociosos de processadores e discos de dados adicionais para atuarem de forma independente no processamento de operadores da consulta. Nossa hipótese é que o uso de nós independentes com recurso de processamento e/ou armazenamento, alocados em uma mesma máquina, pode aumentar a eficiência do processamento paralelo de consultas analíticas em sistemas de gerência de Big Data.

Neste contexto, esta dissertação investiga o impacto da exploração de arquiteturas *multi-core* no processamento paralelo de consultas. Isso é feito por meio do uso de *worker* e *data nodes* alocados em uma mesma máquina. Para avaliar o impacto do uso combinado de *worker nodes* e *data nodes*, utilizamos o mecanismo de execução paralela de consulta relacional MyriaX, componente do sistema de gerência de Big Data Myria (WANG, Jingjing et al., 2017). Esta escolha foi motivada pelo fato deste sistema permitir o uso de *worker nodes* e *data nodes* de forma independente. Desse modo, o MyriaX viabiliza a alocação de *data nodes* e *worker nodes* em uma mesma máquina, apesar de isto nunca ter sido explorado pelo Myria.

Em nossa avaliação experimental, planejamos, executamos e avaliamos os resultados de dois experimentos (Experimento I e II) atacando diferentes pontos cruciais ao desempenho do

processamento paralelo de consultas analíticas. Utilizamos diversas configurações e alocações de *data nodes* e *worker nodes* (cenários) para processamento de consultas em *cluster*, utilizando todo o recurso de processamento (núcleos dos processadores) e armazenamento (disco de dados) disponíveis por máquina. Em ambos os experimentos, o cenário *Baseline* se refere à configuração padrão de nós do MyriaX (quantidade de *data nodes* é igual à quantidade de discos da máquina, e quantidade de *worker nodes* é zero) e o cenário *Avaliação* se refere à configuração de nós com adição de *worker nodes* em uma mesma máquina. Para otimizar a execução destes experimentos, utilizamos um *script* que gerencia as execuções e captura o tempo para 5 rodadas de cada configuração de *data nodes* e *worker nodes*.

O Experimento I avalia o processamento de duas consultas diferentes, uma de Auto-junção e outra de Identificação de Triângulos, sobre uma base de dados da rede social Twitter com grande volume de dados. Este experimento teve como finalidade confirmar a hipótese de que é possível diminuir o tempo de processamento de consultas através da adição de *worker nodes* em núcleos ociosos de processadores. Os resultados deste experimento foram descritos e avaliados em artigo publicado no 32º Simpósio Brasileiro de Banco de Dados (SILVA et al., 2017).

O Experimento II avalia o desempenho do uso de *data nodes* e *worker nodes* no processamento de cargas de trabalhos reais que incluem consultas com diversas características. Para tal, após avaliar um conjunto de benchmarks, selecionamos TPC-DS[6]. A carga de trabalho deste experimento é composta por 11 consultas selecionadas do conjunto de 99 disponíveis para este benchmark. Devido às limitações de infraestrutura do *cluster* utilizado, os cenários de *data nodes* e *worker nodes* foram divididos por quantidade de máquinas e processados sobre bases de dados do TPC-DS com escalas de geração diferentes. Sendo assim, avaliamos os resultados individuais de cada consulta e também da carga de trabalho.

Para o Experimento I, os resultados mostram que sempre há pelo menos um cenário *Avaliação* com melhor desempenho que o cenário *Baseline*. Assim, conseguimos confirmar a hipótese deste experimento. Entretanto, nem sempre a adição gradativa de *worker nodes* apresentou melhores resultados; em quase todos os casos há uma deterioração a partir de um determinado ponto para a consulta de Identificação de Triângulos.

Para o Experimento II, os resultados dividiram a carga de trabalho em dois grupos de consultas: o primeiro grupo obteve significante aceleração no tempo de processamento em ao menos um cenário *Avaliação* e o segundo grupo obteve desaceleração gradativa no tempo de

[6] www.tpc.org/tpcds/

processamento em cenários *Avaliação*. Referente ao grupo com aceleração, todas as consultas apresentaram o comportamento identificado no Experimento I: há uma melhora no desempenho seguida de uma deterioração a partir de um determinado ponto com o acréscimo de *worker nodes*. Os resultados para a carga de trabalho também apresentaram este comportamento em todos os casos.

A contribuição desse trabalho reside nesses dois achados. O primeiro, mostra que é possível obter melhor desempenho, em alguns tipos de consultas, simplesmente adicionando *worker nodes* que usam processadores ociosos de máquinas que já haviam sido alocadas para o processamento da consulta. Esses *worker nodes* atuam no processamento de operadores da consulta que não envolvem leitura de dados, como por exemplo, junções, filtros, etc. O segundo achado mostra que adicionar *worker nodes* indiscriminadamente pode afetar negativamente o desempenho das consultas.

Estes resultados mostram que a característica da consulta precisa ser levada em conta na escolha do melhor cenário de execução, e abre oportunidades para elaboração de heurísticas que possam ser usadas na geração do plano da consulta e na escolha do número de *worker nodes* a ser usado para processar a consulta.

Os próximos capítulos desta dissertação totalizam cinco. O Capítulo 2, nomeado Trabalhos Relacionados, apresenta uma análise das arquiteturas de sistemas de gerência de Big Data. O Capítulo 3, nomeado MyriaX, apresenta uma visão geral do mecanismo de execução paralela de consulta MyriaX do sistema de gerência de Big Data Myria. O Capítulo 4, nomeado Experimento I, apresenta informações sobre o planejamento, execução e avaliação dos experimentos realizados com duas consultas sobre a base do Twitter. O Capítulo 5, nomeado Experimento II, apresenta informações sobre o planejamento, execução e avaliação dos experimentos realizados com carga de trabalho formada por onze consultas do *benchmark* TPC-DS. O Capítulo 6, nomeado Conclusão, apresenta as considerações finais e conclusões sobre os resultados dos experimentos desta dissertação.

CAPÍTULO 2 – SISTEMAS DE GERÊNCIA DE BIG DATA

2.1 INTRODUÇÃO

O crescimento massivo da quantidade de dados disponíveis nas últimas décadas levou ao surgimento de diversos sistemas de gerência de Big Data. Tais sistemas, em sua maioria, utilizam *clusters* ou plataformas de nuvem para distribuição de dados e processamento massivo paralelo de consultas. Alguns destes sistemas oferecem suporte para ambos os ambientes de execução. Entretanto, sendo executados em *cluster* e/ou na nuvem, estes sistemas precisam gerenciar um conjunto de máquinas e distribuir dados e tarefas para que o processamento seja realizado em paralelo.

As arquiteturas utilizadas por sistemas de gerência de Big Data variam. Alguns exigem que cada máquina represente um nó e tenha um disco de dados acoplado (ABOUZEID et al., 2009; ALSUBAIEE et al., 2014; BITTORF et al., 2015; DAS et al., 2013; ISARD et al., 2007; MALEWICZ et al., 2010; MATTOSO et al., 2005; WANG, Jingjing et al., 2017). Outros alocam diversos nós em uma mesma máquina e utilizam de base de dados compartilhada (DAGEVILLE et al., 2016) ou dividem o recurso de armazenamento para cada nó (GUPTA et al., 2015; WARNEKE; KAO, 2009), gerando acesso concorrente em ambos os casos. No entanto, nenhuma destas arquiteturas explora totalmente os recursos de processamento e armazenamento de uma máquina, alocando diversos nós independentes sem gerar acesso concorrente aos discos de dados.

Este capítulo apresenta detalhes de arquiteturas de diversos sistemas de gerência de Big Data, referente ao uso de nós no processamento paralelo de consultas. Os sistemas avaliados incluem Apache Hadoop (Seção 2.2.1), HadoopDB (Seção 2.2.2), Apache Spark (Seção 2.2.3), Asterix (Seção 2.2.4), Impala (Seção 2.2.5), ElasTras (Seção 2.2.6), Pregel (Seção 2.2.7), Dryad (Seção 2.2.8), Vertica (Seção 2.2.9), Presto (Seção 2.2.10), MyriaX (Seção 2.2.16), ParGRES (Seção 2.2.11), Snowflake (Seção 2.2.12), Nephele (Seção 2.2.13), Amazon Redshift (Seção 2.2.14) e Greenplum (Seção 2.2.15).

Sistemas tais como Apache Flink (CARBONE et al., 2015), Storm (TOSHNIWAL et al., 2014), ChronoStream (WU; TAN, 2015), SGuard (KWON et al., 2008), Photon (ANANTHANARAYANAN et al., 2013), Apache Samza[7] e S4 (NEUMEYER et al., 2010) realizam processamento de fluxo de dados (*streaming*). Estes sistemas não realizam leitura de

[7] http://samza.apache.org/

dados em fontes estáticas, portanto, não são considerados junto aos trabalhos relacionados por não efetuarem operações de I/O.

O restante deste capítulo apresenta os detalhes dos Sistemas de Gerência de Big Data (Seção 2.2), discute o uso de nós em Sistemas de Gerência de Big Data (Seção 2.3) e apresenta as considerações finais (Seção 2.4).

2.2 SISTEMAS DE GERÊNCIA DE BIG DATA

Esta seção apresenta detalhes das arquiteturas de sistemas de gerência de Big Data que realizam processamento massivamente paralelo de consultas. Além disso, também é descrito como a consulta é processada em cada arquitetura. Os sistemas abordados nesta seção são: Apache Hadoop, HadoopDB, Apache Spark, Asterix, Impala, ElasTras, Pregel, Dryad, Vertica, Presto, MyriaX, ParGRES, Snowflake, Nephele, Amazon Redshift e Greenplum,

2.2.1 APACHE HADOOP

O Apache Hadoop[8] é um framework que permite processamento distribuído de grandes conjuntos de dados por meio de *cluster* de máquinas usando modelo simples de programação. Este framework foi projetado para escalar a partir de um único servidor para milhares de máquinas, cada uma oferecendo computação e armazenamento local.

O Hadoop é composto pelos seguintes módulos: *Hadoop Common* que contém as bibliotecas necessárias para o funcionamento do framework, o mecanismo *MapReduce* (DEAN; GHEMAWAT, 2004), o sistema de arquivos distribuído Hadoop (HDFS (GHEMAWAT et al., 2003)) e o *Hadoop Yarn* que gerencia os recursos computacionais.

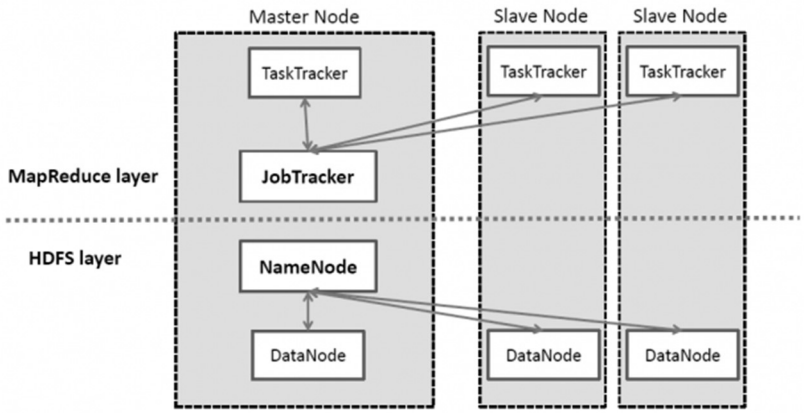

Figura 2.1: Arquitetura Apache Hadoop (BAPPALIGE, 2014)

[8] http://hadoop.apache.org/

A arquitetura de *cluster* do Hadoop inclui um único nó *master* e vários *slave nodes* ou *worker nodes*, como mostra a Figura 2.1. O nó *master* consiste de um *Job Tracker*, *Task Tracker, Name Node* e *Data Node*. Um *worker node* executa funções de *Data Node* e *Task Tracker*.

A execução de consultas no framework *MapReduce* envolve duas etapas: *Map* e *Reduce* (DEAN; GHEMAWAT, 2004). Inicialmente, os dados de entrada são divididos em *M* partes distribuídas entre os *slave nodes*. *M* tarefas *Map* e *R* tarefas *Reduce* são criadas e atribuídas uma para cada *slave node* para processamento paralelo. Um *slave node* que recebe a tarefa *Map* lê o conteúdo correspondente à sua parte e analisa pares de chave/valor que são armazenados em um *buffer* de memória. Periodicamente, este *buffer* é escrito em disco e particionado em *R* partes através de uma função de particionamento (por exemplo *Hash*). A localidade destes pares de chave/valor em disco é passada ao *master node* que repassa aos *slave nodes* responsáveis pela etapa de *Reduce.* Quando um *slave node* que irá realizar a etapa *Reduce* recebe a notificação com a localidades destes pares, ele lê os pares armazenados em disco e realiza a operação necessária. Quando todas as tarefas *Map* e *Reduce* são finalizadas, o *master node* compõe o resultado final e o retorna para o programa do usuário.

2.2.2 HADOOPDB

HadoopDB é um sistema híbrido projetado para fornecer vantagens de um banco de dados paralelo e do framework MapReduce (DEAN; GHEMAWAT, 2004). A ideia básica por trás do HadoopDB é conectar vários sistemas de banco de dados de nó único usando o Hadoop como coordenador de tarefa e estrutura de comunicação de rede (ABOUZEID et al., 2009). Consultas são paralelizadas através de nós usando o framework MapReduce. O HadoopDB alcança tolerância a falhas e habilidade de operar em ambientes heterogêneos herdando a implementação de escalonamento e rastreamento de tarefas do Hadoop. Ele ainda alcança o desempenho de banco de dados paralelos processando consultas dentro do mecanismo de banco de dados.

A Figura 2.2 ilustra a arquitetura do HadoopDB. Como apresenta esta arquitetura, o nó *master* é um único *JobTracker* e nós de trabalho são *TaskTrackers*. O *JobTracker* realiza o escalonamento de tarefas MapReduce em tempo de execução e mantém informação sobre a carga de trabalho de cada *TaskTracker* e sobre os recursos disponíveis. Cada tarefa é processada seguindo o método de processamento do framework *MapReduce* (DEAN; GHEMAWAT, 2004). O *JobTracker* atribui tarefas baseadas na localidade de dado e balanceamento de carga.

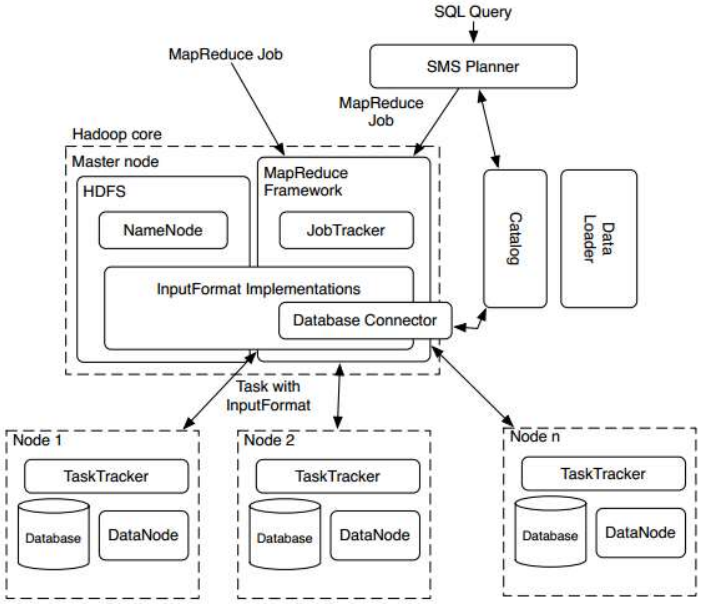

Figura 2.2: Visão geral da arquitetura do HadoopDB (ABOUZEID et al., 2009)

2.2.3 APACHE SPARK

Apache Spark[9] (ZAHARIA et al., 2012) é um sistema que fornece APIs de alto nível em Java, Scala, Python e R. Também suporta um conjunto de ferramentas de alto nível incluindo Spark SQL (ARMBRUST et al., 2015) para SQL e processamento de dados estruturados, MLlib para aprendizagem de máquina, GraphX para processamento de gráfico e Spark Streaming.

Aplicações Spark executam como conjuntos independentes em um cluster, coordenados pelo objeto *SparkContext* no programa principal (chamado de programa *Driver*). Para executar em um cluster, o *SparkContext* pode conectar vários tipos de gerenciadores de cluster (*Mesos*, *Yarn* ou o próprio gerenciador *Standalone Spark*), que alocam recursos através de aplicações.

Uma vez conectado, o Spark instancia um *SparkContext* que adquire *Executors* em máquinas no cluster através do *Cluster Manager*, que são responsáveis por processar e armazenar dados. Na sequência, o Spark envia o código da aplicação (definido por arquivos JAR ou Python passados para o *SparkContext*) para os *Executors*. Finalmente, o *SparkContext* envia tarefas para os *Executors* executarem. Estes retornam resultados de suas operações. A Figura 2.3 ilustra este processo.

[9] https://spark.apache.org/

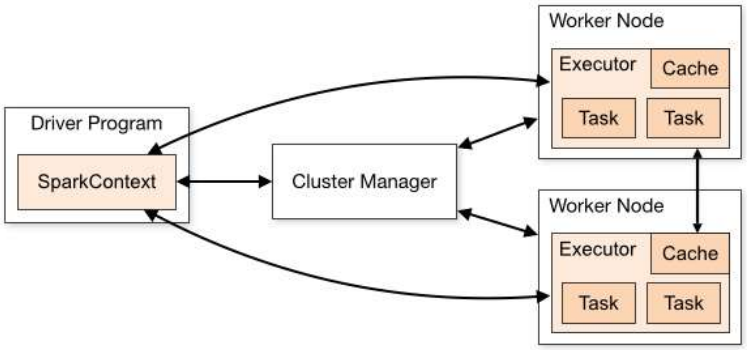

Figura 2.3: Visão geral Apache Spark. FONTE: https://spark.apache.org/docs/latest/cluster-overview.html

Na arquitetura do Apache Spark cada aplicação recebe seu conjunto de *Executors*, que o mantém ativo durante toda a execução da aplicação e processa tarefas em várias *threads*. Tal ação gera o benefício de isolamento de aplicações uma da outra, tanto do lado do escalonador (cada *Driver* escalona suas tarefas) quanto do lado do *Executor* (tarefas a partir de aplicações diferentes executam em JVMs diferentes). Contudo, isto significa que dados não podem ser compartilhados através de diferentes aplicações Spark (instâncias de *SparkContext*) sem escrever em uma fonte externa de armazenamento.

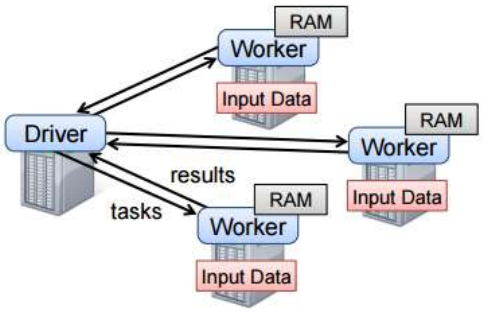

Figura 2.4: Arquitetura de *cluster* Apache Spark (ZAHARIA et al., 2012)

2.2.4 ASTERIX

O Asterix (ALSUBAIEE et al., 2014) é caracterizado como uma junção entre plataformas analíticas de Big Data, RDBMS paralelo e armazenamento NoSQL. Este sistema oferece armazenamento nativo e indexação de dados, bem como consulta de *datasets* em HDFS ou arquivos locais. Isto permite eficiência para pequenas e grandes consultas. O Asterix[10] tem um modelo de dados aberto que lida com dados aninhados complexos, bem como dados planos e casos de uso variando de "*schema first*" para "*schema never*". O Asterix tem uma linguagem

[10] https://asterixdb.apache.org/

de consulta, a AQL (*Asterix Query Language*), que suporta consulta declarativa sobre muitos conjuntos de dados.

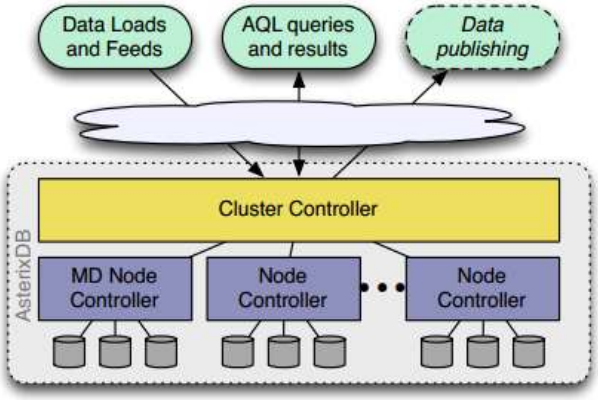

Figura 2.5: Visão geral Asterix (ALSUBAIEE et al., 2014)

A Figura 2.5 fornece uma visão geral do Asterix e sua arquitetura lógica. Ele possui entrada de dados através de carregamento, alimentação contínua e/ou operações de inserção. Os dados são acessados via consulta (de forma síncrona ou assíncrona). O *Cluster Controller* é o ponto de entrada lógico das requisições de usuários; o *Node Controller* e *MD* (*Meta Data*) *Node Controller* fornecem acesso para metadados do Asterix e agregam poder de processamento do *cluster shared-nothing* subjacente.

Para executar uma tarefa, o Asterix utiliza o mecanismo Hyracks (BORKAR et al., 2011). Hyracks é um mecanismo responsável por aceitar e gerenciar requisições de computação de dados em paralelo, enviadas ou por uma das camadas acima da estrutura de execução de consulta do Asterix ou diretamente pelo usuário final do Hyracks. Tarefas são submetidas ao Hyracks em forma de DAGs (*Directed Acyclic Graphs*) constituídos de *Operators* e *Conectors*. *Operators* são responsáveis por consumir partições de entrada e produzir partições de saída, enquanto *Conectors* redistribuem dados das partições de saída e fornecem partições de entrada para o próximo operador. O grau de paralelismo para estes operadores é o número de partições que são usadas para armazenar o *Dataset*. Estes *Datasets* são armazenados nos nós do *cluster* em forma de árvores B^+ com particionamento baseado em *hash* através da chave primária.

Para processar a consulta, o Asterix compila o código AQL num programa algébrico *Algebricks*. Este programa é otimizado através de regras de reescrita algébrica que reordenam os operadores *Algebricks* e introduzem paralelismo para execução escalável. Após a geração do código de otimização, o plano de consulta físico resultante é traduzido em uma tarefa que usa o Hyracks para processar o resultado de consulta desejado.

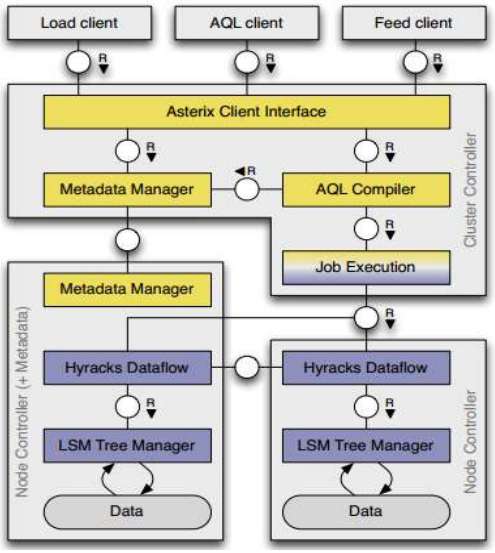

Figura 2.6: Visão geral Hyracks no Asterix (ALSUBAIEE et al., 2014)

Como mostra a Figura 2.6, para executar uma tarefa, o Hyracks analisa o grafo de atividades produzido para identificar a coleção de atividades que podem ser executadas em qualquer tempo. Cada estágio é paralelizado e o grafo resultante é executado na ordem de dependências das subtarefas, ou seja, cada subtarefa é processada quando as atividades necessárias para seu processamento são concluídas por subtarefas das quais ela depende.

2.2.5 IMPALA

O Impala é um mecanismo de processamento de consultas SQL massivamente paralelo (MPP – *Massive Parallel Processing*) de código aberto (BITTORF et al., 2015). O objetivo deste mecanismo é combinar suporte SQL e desempenho multiusuário de um banco de dados analítico tradicional com a escalabilidade e flexibilidade do Apache Hadoop[8], além de extensões de segurança e gerência da Cloudera Enterprise.

Para reduzir latência de rede, tal como a que ocorre a partir da utilização de *MapReduce* ou pela leitura de dados remotamente, o Impala implementa uma arquitetura distribuída baseada em processos *Daemon*, que são responsáveis por todos os aspectos de execução da consulta e que executam nas mesmas máquinas com o resto da infraestrutura Hadoop.

O Impala executa em um *cluster* Hadoop. Ele é implantado a partir de um mecanismo de armazenamento subjacente, diferente de sistemas gerenciadores de banco de dados relacionais tradicionais, onde o processamento da consulta e o mecanismo de armazenamento subjacente são componentes de um único sistema fortemente acoplado. A visão geral da arquitetura do Impala é ilustrada na Figura 2.7.

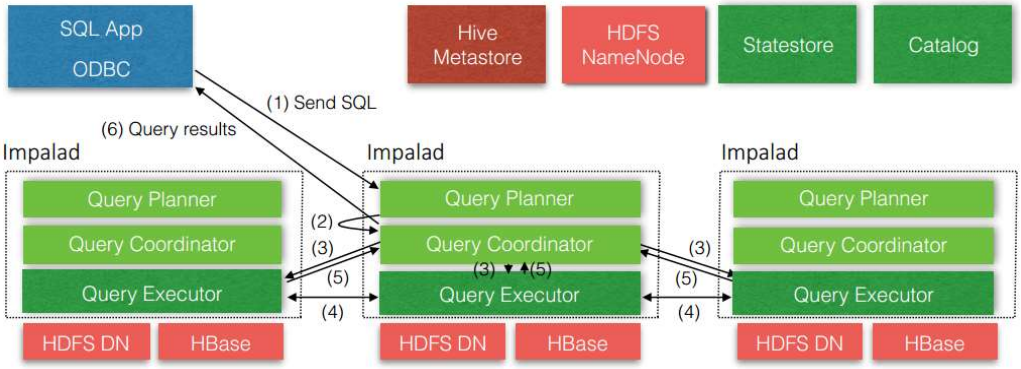

Figura 2.7: Visão geral da arquitetura do Impala (BITTORF et al., 2015)

Uma implantação do Impala é composta de três serviços. O serviço *Impala Daemon* (*Impalad*) é responsável por aceitar consultas a partir de processos clientes e orquestrar suas execuções no *cluster*, e pela execução de fragmentos individuais de consulta em nome de outros *Impalad*. Quando um *Impalad* opera primeiro no papel de gerente da execução da consulta, ele se torna o coordenador desta consulta.

Um *Impalad* é implantado em qualquer máquina no cluster que está executando um *DataNode*. Isto permite ao Impala tomar vantagem da localidade do dado, e ler blocos a partir do sistema de arquivos sem uso de rede.

O processo de compilação de consultas no Impala segue o tradicional: análise sintática (*parsing*), análise semântica e planejamento/otimização da consulta. Um plano de consulta executável é construído em duas fases: (1) planejamento e (2) fragmentação e paralelização do plano. Em (1) a consulta é traduzida em um plano não executável. Em (2) um plano de execução distribuído é produzido a partir do plano não executável. O modelo de execução é o tradicional Volcano-style (GRAEFE, 1990).

2.2.6 ELASTRAS

O ElasTras é um banco de dados relacional transacional e elástico (elasticidade de recursos) para plataforma de nuvem. Ele é escalável para se adaptar a cargas variantes, mas também é tolerante a falhas e auto gerenciável para eliminar sobrecargas administrativas frequentemente atribuídas a grandes instalações de RDBMSs (DAS et al., 2013).

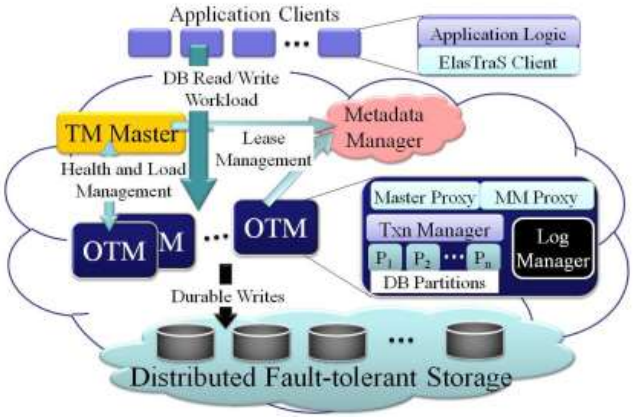

Figura 2.8: Visão geral da arquitetura do ElasTras (DAS et al., 2013)

A Figura 2.8 fornece uma visão geral de alto nível da arquitetura do ElasTras. Na parte inferior está o armazenamento distribuído tolerante a falha (*Distributed Fault-tolerant Storage* ou DFS) que funciona como um armazenamento permanente. No coração do sistema estão os *Gestores de Transação* (OTM) que possuem uma ou mais partições e fornecem garantias transacionais. O *TM Master* do sistema é responsável por monitorar os OTMs, recuperar nós falhos e executar balanceamento de carga e escalonamento elástico. O *Gestor de Metadados* (*MM*) mantém o estado do sistema e o mapeamento de partições para OTMs. Há também o *client library* (biblioteca do cliente), projetada para esconder a complexidade da localização de partições na presença de vários sistemas dinâmicos.

No ElasTras, quando uma consulta é recebida pela aplicação do cliente, ela é despachada em paralelo para todas os OTMs que contêm partições de interesse desta consulta. Os OTMs executam as consultas de forma independente retornando os resultados para a aplicação do cliente que mescla os resultados antes de retorná-los ao cliente. O ElasTras utiliza o DFS como gerenciador de armazenamento distribuído tolerante a falha sob uma estrutura de armazenamento, tal como HDFS.

2.2.7 PREGEL

O Pregel é um framework tolerante a falhas para processamento distribuído de grafos em larga escala (MALEWICZ et al., 2010) baseado no framework MapReduce (DEAN; GHEMAWAT, 2004). A entrada para computações no Pregel é um grafo direcionado em que cada vértice é unicamente identificado por um conjunto de caracteres chamado *vertex identifier*. Uma tarefa Pregel é constituída de uma entrada, onde o grafo é inicializado, seguido de uma sequência de *supersteps* (conjunto de iterações) separados pelos pontos globais de

sincronização até que o algoritmo termine e finalize com uma saída. A saída é o conjunto de valores de saída dos vértices.

Durante um *superstep* S o framework invoca uma função definida pelo usuário para cada vértice do grafo, em paralelo. Ela poder ler mensagens enviadas para um vértice V no *superstep* S – 1 (*superstep* anterior), enviar mensagens para outros vértices que irão recebê-las em S + 1 (*superstep* seguinte), e modificar o estado de V e suas arestas de saída. Estas mensagens são especificadas pelo usuário como um parâmetro do modelo (por exemplo, uma classe vértice) e consistem de um valor e o nome do vértice de destino.

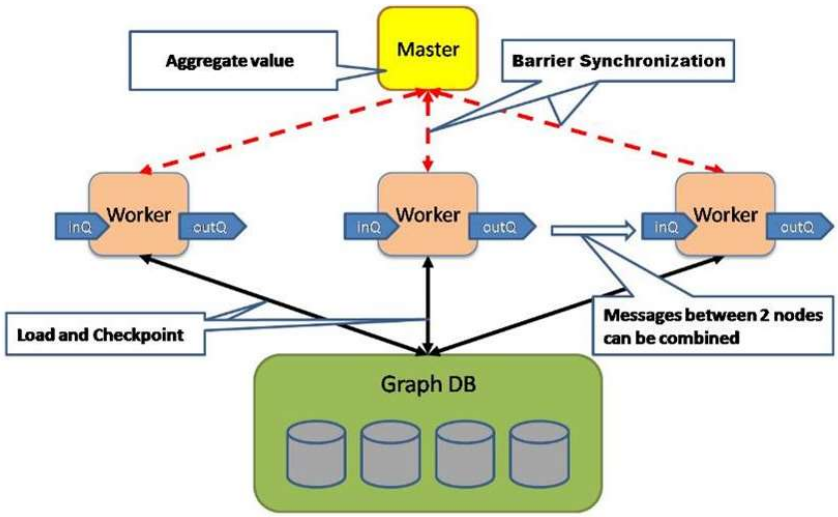

Figura 2.9: Visão geral da arquitetura do Pregel (HO, 2010)

O Pregel é executado em um sistema de gerência de cluster que escalona tarefas para otimizar alocação de recursos. A persistência de dados é realizada em arquivos em um sistema de armazenamento distribuído, e dados temporários são armazenados em disco local. A visão geral da arquitetura do sistema Pregel é apresentada na Figura 2.9. A execução de uma tarefa Pregel consiste de vários estágios:

(I) Muitas cópias de uma tarefa iniciam executando em um *cluster* de máquinas. Uma destas cópias atua como *master*. O *master* não recebe qualquer parte do grafo, mas é responsável pela coordenação das atividades das máquinas *workers*. Os *workers* usam o sistema de gerência do cluster para identificar o *master* e enviar mensagens de registro.

(II) O *master* determina quantas partições o grafo irá ter, e atribui uma ou mais partições para cada *worker*. Cada *worker* é responsável por manter o estado destas partições dos grafos, processar os vértices e gerenciar o envio e recebimento de mensagens de outros *workers*. Cada *worker* recebe o conjunto completo de atribuições para todos os *workers*.

(III) O *master* atribui uma porção da entrada do usuário para cada *worker*. A entrada é tratada como um conjunto de registros. Cada um contém um número arbitrário de vértices e arestas. Após o carregamento da entrada ser finalizado, todos os vértices são marcados como ativos.

(IV) O *master* instrui cada *worker* para executar um *superstep*. O *worker* realiza processamento através dos vértices ativos, usando uma *thread* para cada partição. Quando o *worker* é finalizado ele responde ao *master*, informando quantos vértices estarão ativos no próximo *superstep*.

(V) Após o fim do processamento, o *master* pode instruir cada *worker* para salvar suas porções do grafo.

2.2.8 DRYAD

Dryad é um mecanismo de execução distribuído de alto desempenho e propósito geral (ISARD et al., 2007). Este sistema resolve diversos problemas críticos de uma aplicação concorrente e distribuída: escalonamento de recursos, controle de concorrência e recuperação de falhas de computadores ou de comunicação.

A estrutura geral de uma tarefa Dryad é determinada pelo seu fluxo de comunicação. Uma tarefa é um DAG onde cada vértice é uma subtarefa e cada aresta representa canais de dados. Tal grafo é mapeado sobre recursos físicos em tempo de execução. Ainda em tempo de execução, cada canal é usado para transportar uma sequência finita de itens estruturados. Esta abstração de canal tem várias implementações concretas que usam memória compartilhada, canais TCP ou arquivos temporários persistidos em um sistema de arquivos. Na medida em que a subtarefa de cada vértice é atribuída, os canais produzem e consomem objetos.

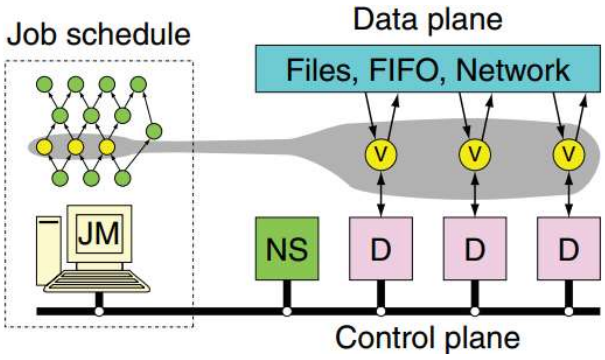

Figura 2.10: Visão geral da arquitetura do Dryad (ISARD et al., 2007)

A arquitetura do sistema Dryad é apresentada na Figura 2.10. Uma tarefa Dryad é coordenada por um processo chamado *Job Manager* (JM) que executa dentro do cluster ou em uma estação de trabalho do usuário com acesso de rede ao *cluster*. O JM contém o código

específico da aplicação para construir o grafo de comunicação da tarefa juntamente com o código da biblioteca para escaloná-la através dos recursos disponíveis. Todos os dados são enviados diretamente entre vértices e, assim, o JM é responsável apenas por controlar decisões.

O *cluster* tem um *Name Server* (NS) que pode ser usado para enumerar todos os computadores disponíveis. O NS também expõe a posição de cada computador dentro da topologia de rede e, então, as decisões de escalonamento podem contar com a localidade. Existem *Daemons* (D) executando em cada computador a partir da memória *cache*.

Quanto à persistência, o Dryad utiliza um sistema de armazenamento distribuído, não descrito no artigo que descreve o sistema, que compartilha com o *Google File System* (GHEMAWAT et al., 2003) a propriedade de que grandes arquivos podem ser quebrados em pequenas partes que são distribuídas e replicadas através de discos locais no cluster de computadores, assim como o HDFS.

2.2.9 VERTICA

O Banco de Dados Analítico Vertica é um sistema distribuído e massivamente paralelo (MPP) (LAMB et al., 2012). Neste sistema, a arquitetura física é projetada para distribuir armazenamento físico e permitir execução de consulta paralela sobre uma grande coleção de recursos computacionais. Os componentes desta arquitetura são: *Host, Instance, Node, Cluster* e *Database*.

Host é um sistema de computador com processador, memória RAM, disco local e interface de rede TCP/IP. *Hosts* não compartilham disco local nem memória RAM entre si. *Instance* consiste de um processo Vertica e armazenamento de disco executando em um *Host.* Apenas um *Instance* do Vertica pode executar em cada *host*. *Node* é um *host* configurado para executar um *Instance* do Vertica. Ele é um membro do *Database Cluster*. *Cluster* refere-se à coleção de *Hosts* ligados ao banco de dados. *Database* é um *Cluster* de nós que, quando ativo, pode executar armazenamento de dados distribuído e instruções SQL através de interfaces de usuário programáticas, interativas e administrativas.

Quando uma consulta é submetida no Vertica, o *Initiator* (nó específico em que a conexão foi realizada) otimiza e planeja a execução da consulta. Este planejamento resulta em um Plano Detalhado. Este plano mapeia as etapas que constituem o processamento da consulta. Ele pode ser visualizado no Console de Gerência. O otimizador do Vertica então divide o Plano Detalhado em pequenos planos distribuídos que são atribuídos para *executor nodes* (qualquer nó que participa da execução de uma instrução SQL específica). Na etapa final do Plano Detalhado, o nó *Initiator* combina resultados em operação de agrupamento, mescla vários

conjuntos de resultados parciais ordenados de todos os nós *Executors*, formata o resultado e retorna ao cliente. A Figura 2.11 apresenta a arquitetura do Console de Gerência do Vertica.

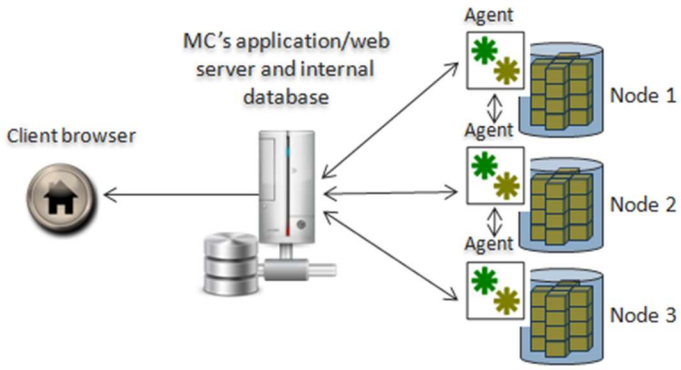

Figura 2.11: Arquitetura do Console de Gerência do Vertica (VERTICA, 2017)

2.2.10 PRESTO

O Presto[11] é um mecanismo de consulta SQL distribuído de código aberto para execução de consultas analíticas interativas contra fontes de dados de todos os tamanhos, variando de gigabytes até terabytes. Este sistema foi projetado como uma alternativa para ferramentas que consultam HDFS e uma única consulta pode combinar dados de múltiplas fontes, e.g. Hive (THUSOO et al., 2009), Cassandra, bancos de dados relacionais ou mesmo bases de dados proprietárias, não se limitando ao HDFS. Este sistema oferece conexão com o *benchmark* TPC-H[12]. Isto permite ao usuário gerar dados automaticamente para execução de testes com esta *workload*.

O sistema Presto é distribuído e executa em um cluster de máquinas. A instalação completa inclui um *Coordinator* (coordenador) e vários *Workers* (nós trabalhadores). Consultas são submetidas a partir de aplicações cliente para o coordenador. O coordenador analisa, planeja a execução da consulta e distribui o processamento para os *Workers*. Estes realizam a *pipeline* de processamento e retornam os resultados ao *Coordinator*. A Figura 2.12 ilustra a arquitetura deste sistema.

[11] http://prestodb.io/
[12] http://www.tpc.org/tpch/

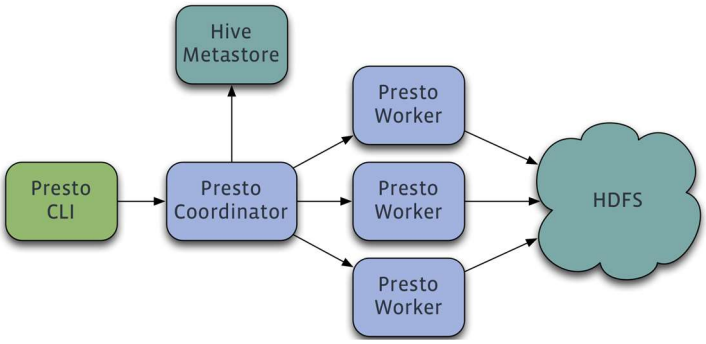

Figura 2.12: Arquitetura do Presto. FONTE: https://prestodb.io/overview.html

2.2.11 PARGRES

O ParGRES é um *middleware* que atua entre a aplicação do cliente e SGBDs (Sistemas de Gerência de Banco de Dados). Ele é capaz de gerenciar operações em *cluster* de máquinas de propósito geral (MATTOSO et al., 2005). O ParGRES combina diferentes técnicas de processamento paralelo de consultas, além de processar requisições de atualizações na base de dados e balanceamento de carga. A comunicação entre o ParGRES e o SGBD é baseada no padrão SQL, o que acrescenta flexibilidade na escolha do SGBD a ser utilizado pela aplicação cliente.

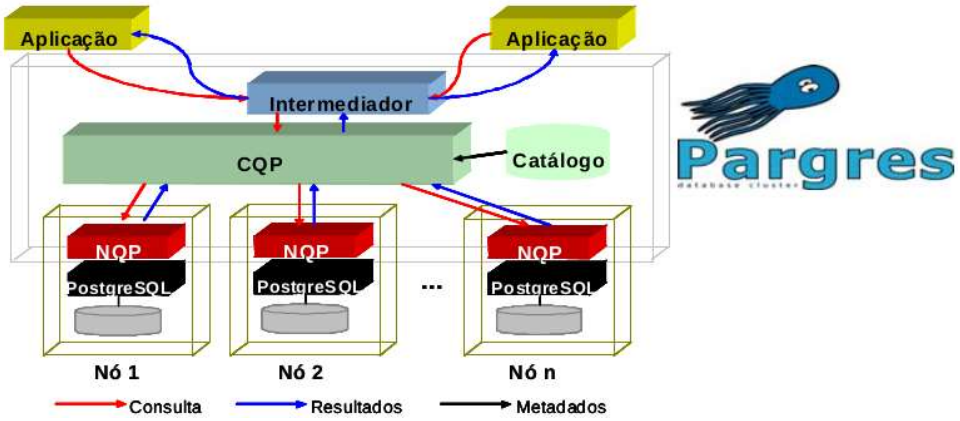

Figura 2.13: Arquitetura do ParGRES (MATTOSO et al., 2005)

O ParGRES possui uma arquitetura descentralizada (MATTOSO et al., 2005), como apresenta a Figura 2.13, com dois tipos de componentes: globais e locais. Componentes globais (Intermediador e o Processador de Consultas de *Cluster* (CQP - *Cluster Query Processor*)) executam tarefas que envolvem vários nós do *cluster*. O CQP é responsável por coordenar os demais componentes e controlar a execução das requisições do cliente, que são enviadas/recebidas pelo Intermediador. Componentes locais (Processador de Consultas de Nó (NQP - *Node Query Processor*) e o SGBD) executam tarefas em apenas um nó. Na versão atual do sistema, o SGBD utilizado é o PostgreSQL.

As principais tarefas executas pelo ParGRES são: (i) tradução da consulta SQL para execução paralela, (ii) processamento de consultas com paralelismo inter/intra-consulta, (iii) composição de resultados e (iv) processamento de atualizações (MATTOSO et al., 2005). O paralelismo intra-consulta decompõe uma consulta complexa em subconsultas para execução em paralelo, onde cada subconsulta executa em um fragmento de dados diferente (HARDAVELLAS; PANDIS, 2009b). O paralelismo inter-consulta executa consultas distintas de forma concorrente em nós do cluster (HARDAVELLAS; PANDIS, 2009a).

O CQP possui um tradutor que contém um analisador sintático para processar requisições do cliente e efetuar a fragmentação virtual da base de dados. A fragmentação virtual consiste em cada nó acessar apenas parte de uma base que está totalmente replicada em cada nó. O tradutor identifica atributos e relações envolvidas na consulta e o CQP decide qual atributo será utilizado na fragmentação virtual.

A tarefa (i) é realizada pelo CQP que efetua a análise das consultas para decidir qual tipo de paralelismo será utilizado em cada uma, bem como os nós que farão parte deste processamento. Para tal, utiliza informações presentes no Catálogo. Este, por sua vez, armazena informações simples inerentes à implementação da fragmentação virtual adaptativa (MATTOSO et al., 2005).

No paralelismo inter-consulta, o CQP envia a consulta ao NQP do nó com o menor número de tarefas pendentes. O NQP repassa a consulta ao SGBD, que a processa. O resultado é encaminhado até a aplicação do cliente. No paralelismo intra-consulta, o CQP aloca os NQPs e encaminha a cada um uma sub-consulta para execução local. Cada NQP processa sua sub-consulta e encaminha os resultados ao CQP para composição do resultado final da consulta. Após o CQP receber todos os resultados de todos os NQPs, ele compõe o resultado final e o encaminha à aplicação do cliente.

2.2.12 SNOWFLAKE

O Snowflake é um sistema elástico, altamente escalável, transacional, multicliente, com total suporte SQL e extensões internas para dados semiestruturados e sem esquema (DAGEVILLE et al., 2016). Ao contrário de outros sistemas gerenciadores de dados na nuvem, o Snowflake não é baseado em Hadoop, PostgreSQL ou algum sistema semelhante.

O Snowflake tem uma arquitetura orientada a serviços escaláveis independentes e tolerante a falhas. Estes serviços são compostos de três estruturas (Figura 2.14): *Data Storage*, *Virtual Warehouses* e *Cloud Services*.

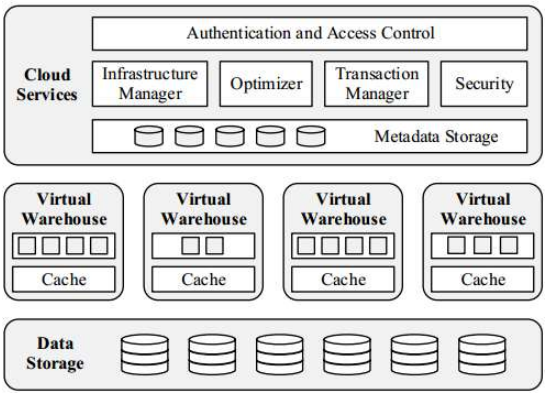

Figura 2.14: Visão geral da arquitetura do Snowflake (DAGEVILLE et al., 2016)

Como *Data Storage*, o Snowflake utiliza a plataforma Amazon Web Services (AWS) e o serviço de armazenamento S3. As tabelas são particionadas horizontalmente em arquivos grandes e imutáveis, equivalentes a blocos ou páginas em um sistema de banco de dados tradicional. Cada arquivo de tabela tem um cabeçalho que, entre outros metadados, contém os detalhes de cada coluna dentro do arquivo. Como o S3 permite requisições GET sobre partes de arquivos, o processador de consultas precisa apenas baixar o cabeçalho dos arquivos e as colunas de interesse.

A estrutura de *Virtual Warehouses* consiste de *clusters* de instâncias EC2. Cada um desses clusters é apresentado ao seu único usuário através de uma abstração chamada depósito de dados virtual (*Virtual Warehouses* ou VW). As instâncias individuais EC2 que compõe um VW são chamados de *Worker Nodes*.

A estrutura de *Cloud Services* é fortemente multicliente. Isto melhora a utilização e reduz cargas administrativas, o que permite melhor economia de escala em relação a arquiteturas tradicionais onde todos os usuários têm uma visão completamente privada do sistema.

O otimizador (*Optimizer*) de consulta do Snowflake utiliza a abordagem de otimização baseada em custo. Aqui, todos os estágios iniciais do clico de vida de consulta são controlados: análise, resolução de objeto, controle de acesso e otimização de plano. Uma vez que o otimizador completa a otimização do plano de consulta, o plano de execução resultante é distribuído para todos os *Worker Nodes* que participam do processamento da consulta. Quando o processamento é concluído pelos *Worker Nodes*, o resultado é encaminhado de volta até a aplicação do cliente.

2.2.13 NEPHELE

O Nephele é um framework de processamento massivamente paralelo projetado explicitamente para ambientes em nuvem (WARNEKE; KAO, 2009). Mais notavelmente, Nephele é o primeiro framework de processamento de dados que inclui a possibilidade de alocar/desalocar diferentes recursos computacionais na nuvem durante o escalonamento e duração da execução da tarefa. O Nephele segue a arquitetura clássica *master-worker* ilustrada na Figura 2.15.

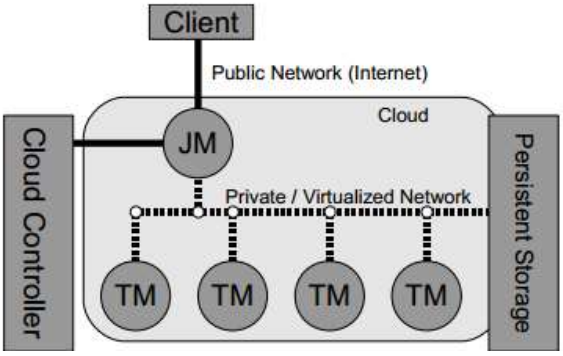

Figura 2.15: Visão geral da arquitetura do Nephele (WARNEKE; KAO, 2009)

Antes da submissão de uma tarefa para o Nephele, o usuário deve iniciar uma instância do *Job Manager* (JM) na nuvem. O JM recebe as tarefas do cliente e é responsável pelo escalonamento e coordenação da execução destas. Ele é capaz de se comunicar com o *Cloud Controller* (Controlador de Nuvem) através de uma interface de serviço web e pode alocar ou desalocar máquinas virtuais de acordo com a fase de execução da tarefa atual.

O Nephele tem como entrada um programa PACT (*Parallelization Contracts* ou Contratos de Paralelização). O modelo de programação PACT é uma generalização do MapReduce (DEAN; GHEMAWAT, 2004) baseado no modelo de dados chave/valor (ALEXANDROV et al., 2010). Tratando-se de uma tarefa, a execução do programa PACT é entendida pelo Nephele como sendo um DAG, onde vértices representam subtarefas e arestas representam canais de comunicação. Contudo, o DAG inicial não representa a execução paralela. Antes de iniciar a execução, o Nephele gera o grafo de fluxo de dados paralelo a partir do DAG, onde os vértices são multiplicados de acordo com o grau de paralelismo desejado.

Cada tarefa Nephele é executada por um conjunto de instâncias. Cada instância executa em um componente local do framework Nephele chamado *Task Manager* (TM). Um TM recebe uma ou mais tarefas do JM em um momento, as executa e informa ao JM sobre o fim da tarefa ou possíveis erros. No recebimento de mensagens o JM decide, dependendo da tarefa, quantas

e quais tipos de instâncias devem ser alocadas/desalocadas para mensurar o processamento contínuo baseado em custo/benefício.

O sistema de armazenamento da nuvem (e.g., Amazon S3) é utilizado para armazenar as entradas de dados das tarefas e eventuais saídas de dados recebidas. Ele deve ser acessível para ambos o JM e o conjunto de TM.

2.2.14 AMAZON REDSHIFT

O Amazon RedShift é um sistema de gerência de banco de dados e processamento de consulta, massivamente paralelo, projetado para suportar cargas de trabalhos analíticas (GUPTA et al., 2015). A arquitetura do Amazon Redshift consiste de um mecanismo de banco de dados que fornece armazenamento de dados e execução SQL (*Data Plane*), e um mecanismo que um *workflow* para monitorar e gerenciar o banco de dados (*Control Plane*), e outros serviços web Amazon para suportar a execução do *Data Plane* e *Control Plane*.

A visão geral da arquitetura do sistema Amazon Redshift (Figura 2.16) pode ser vista como a integração da arquitetura tradicional de banco de dados relacional distribuída e paralela com arquitetura orientada a serviço.

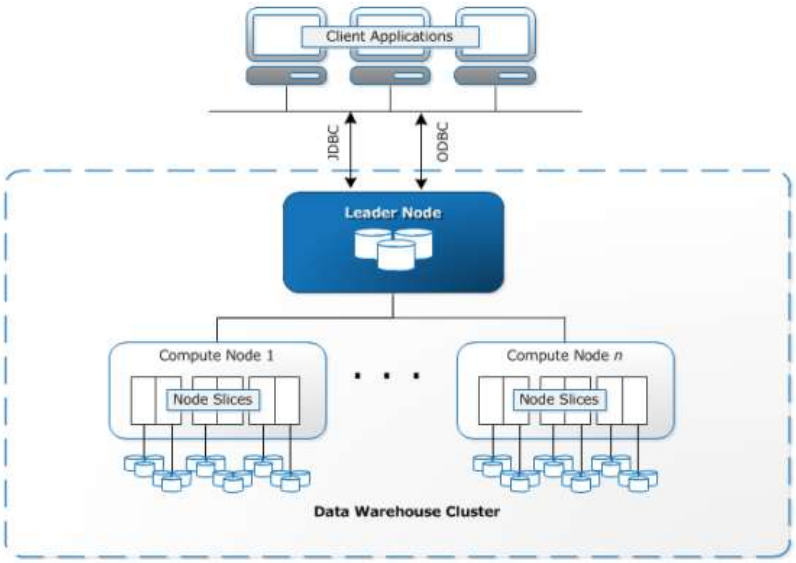

Figura 2.16: Visão geral da arquitetura do Amazon Redshift (GUPTA et al., 2015)

O armazenamento e processamento de dados é distribuído através de um ou mais nós, que fornecem escalabilidade quase linear para manutenção e consulta de conjuntos de dados. Blocos de dados são replicados dentro de cada instância do *cluster* e dentro do serviço de armazenamento simples da Amazon (S3), como estratégia de recuperação em caso de falhas.

Um *cluster* Amazon Redshift é compreendido de um *leader node* (nó líder) e um ou mais *compute nodes* (nós de computação). O *leader node* aceita requisições de clientes, gera e

compila planos de consultas para execução em nós de computação, executa agregação final de resultados quando requerido e coordena serialização e estado de transações. Os *compute nodes* executam o trabalho pesado inerente ao processamento de consultas e manipulação de dados locais.

Um *compute node* é particionado em fatias; uma fatia para cada núcleo do processador *multi-core* dos nós. Para cada fatia é alocada uma porção de memória e espaço em disco, onde uma porção da carga de trabalho atribuída para o nó é processada. Dados armazenados em tabelas são automaticamente distribuídos através de *compute nodes*, para permitir escalar para grandes conjuntos de dados, e entre fatias dentro de cada *compute node*, para reduzir contenção através do paralelismo intra-máquina utilizando núcleos de processamento.

O processamento de consultas dentro do Amazon Redshift inicia com a geração do plano de consulta e compilação para C++ e código de máquina no *leader node*. Os parâmetros e planos executáveis são enviados para cada *compute node* participante na consulta. Nos *compute nodes*, os planos executáveis são processados e os resultados intermediários são enviados de volta para o *leader node* para agregação final. Cada fatia no *compute node* pode executar várias operações, tais como leitura, filtro, processar junções etc., em paralelo.

2.2.15 GREENPLUM

O Greenplum Database[13] é um servidor de banco de dados de processamento massivamente paralelo. A arquitetura deste sistema é projetada especialmente para gerenciar armazéns de dados analíticos de larga escala e cargas de trabalho de inteligência de negócios. O Greenplum é baseado na tecnologia de código aberto PostgreSQL.

Este sistema, essencialmente, se resume a várias instâncias de banco de dados PostgreSQL agindo juntas. A parte interna do PostgreSQL foi modificada para suportar a estrutura paralela do Greenplum. Os componentes de catálogo do sistema, otimizador, executor de consulta e gerenciador de transação foram modificados e melhorados para serem capazes de executar consultas simultâneas por meio de todas as instâncias de banco de dados PostgreSQL paralelas. Este sistema também permite que o usuário realize configurações para alocação de recursos durante a execução de *workloads*.

O Greenplum armazena e processa grandes quantidades de dados distribuindo-os e processando cargas de trabalho através de várias máquinas. O *master* é o ponto de entrada para

[13] http://greenplum.org/

o sistema de banco de dados Greeplum. A visão geral da arquitetura do Greenplum é ilustrada na Figura 2.17.

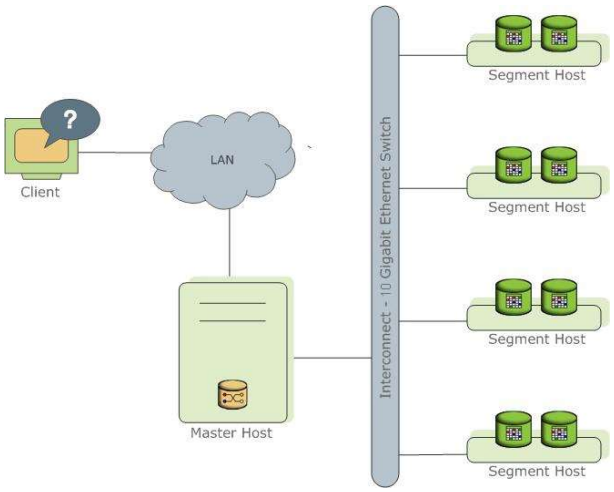

Figura 2.17: Visão geral da arquitetura do Greenplum (PIVOTAL SOFTWARE, 2017)

O *Master* é a entrada do sistema de banco de dados Greenplum, aceitando conexões de clientes e consultas SQL, distribuindo tarefas para instâncias *segment*. É nele que reside o catálogo global do sistema. O *master* autentica conexões de clientes, processa comandos SQL de entrada, coordena os resultados retornados pelos *segments* e apresenta o resultado final para o programa cliente.

Instâncias *segment* são banco de dados PostgreSQL independentes que armazenam uma porção dos dados e executam a maioria do processamento da consulta. Estas instâncias executam em servidores chamados *segment hosts*. Um *segment host* executa de dois a oito *segments* Greenplum, dependendo dos núcleos de CPU, RAM, armazenamento, interfaces de rede e cargas de trabalho. Eles têm configurações idênticas. A chave para obter o melhor desempenho do banco de dados Greenplum é distribuir dados e cargas de trabalho uniformemente através de um grande número de *segments* igualmente capazes, onde todos os *segments* iniciam e finalizam sua tarefa simultaneamente.

O Greenplum contém um otimizador de consulta chamado Orca (SOLIMAN et al., 2014), que foi projetado para processamento de cargas de trabalho complexas. Dentre as principais características deste otimizador se destacam a modularidade, que permite que ele seja portado para outros sistemas de gerenciamento de dados, a extensibilidade, pois ele representa todos os elementos de uma consulta e sua otimização como cidadãos de primeira classe em pé de igualdade, o uso da tecnologia *multi-core*, pois ele distribui subtarefas individuais de otimização para múltiplos núcleos de processamento, a verificabilidade, pois ele verifica

correções e desempenho em mecanismos internos, e finalmente o desempenho, pois ele oferece *speeedup* de 10x até 100x.

O Greenplum cria vários processos para realizar o processamento de uma consulta. No *master*, existe um processo chamado de *query dispatcher* (QD) responsável por criar e despachar o plano de consulta para os *segments*. Nos *segments*, existe um processo chamado *query executor* (QE) responsável por executar uma parcela da tarefa e comunicar os resultados intermediários para outros processos QE. Após a conclusão da tarefa o *master* acumula todos os resultados recebidos pelos *segments* e apresenta o resultado final para a aplicação do cliente.

2.2.16 MYRIAX

O MyriaX é um mecanismo de execução de consultas relacional, *shared-nothing* e paralelo do Serviço de Gerência de Big Data Myria(WANG, Jingjing et al., 2017). Sua arquitetura é composta por um *master node*, responsável por gerenciar os demais nós; por *worker nodes*, responsáveis pela execução de operadores de álgebra relacional durante o *pipeline* do processamento de consulta (ordenação, agregação, junção, etc.); e *data nodes*, responsáveis por armazenar partições e réplicas de tabelas do banco de dados (HALPERIN et al., 2014).

O MyriaX gerencia cada nó como sendo de um único tipo que possui recurso de processamento e armazenamento (*data node*). A diferenciação entre *data node* e *worker node* acontece na atribuição de tarefas para cada nó. Nós que recebem operações de leitura e/ou escrita de dados em disco atuam como *data nodes*. Por outro lado, nós que não recebem tarefas de leitura e/ou escrita de dados, mas apenas de processamento, atuam como *worker nodes*. *Data nodes* podem também participar como *worker nodes* no processamento de consultas pois, ao final das operações de leitura dos dados (*scan*), o processador do nó pode ser utilizado para processamento de operadores da consulta. A Figura 2.18 ilustra uma implantação do MyriaX com nove máquinas, sendo 8 nós (*data nodes*), além do *master node*.

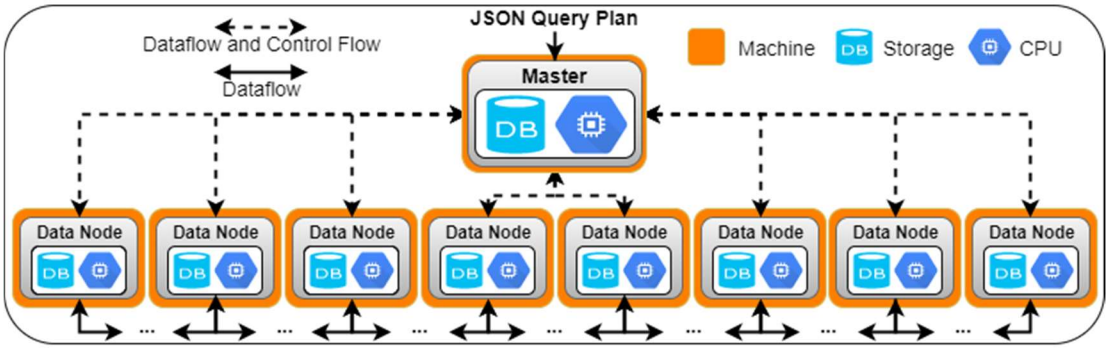

Figura 2.18: Visão geral da arquitetura do MyriaX.

O plano de consulta da tarefa é gerado e passado ao MyriaX através de algum otimizador de consultas, tal como o RACO (WANG, Jingjing et al., 2017). A execução acontece em paralelo, com múltiplos fragmentos independentes ou encadeados (*pipelined*) entre nós, no caso comum do paralelismo de dados na avaliação da consulta. A partir da descrição do plano da consulta (em formato JSON), o MyriaX atribui cada fragmento ao seu conjunto de nós, tal como foram especificados, para processamento dos operadores. Cada fragmento inicia seu processamento a partir do momento que o anterior conclui parte de sua tarefa, em um *pipeline* de operações de álgebra relacional. Caso não haja fragmentos anteriores, como no caso de operações de leitura de dados, o *master node* inicia a execução do fragmento imediatamente. Mais informações sobre o MyriaX, plataforma base dos experimentos realizados nessa dissertação, são fornecidas no Capítulo 3.

2.3 USO DE NÓS EM SISTEMAS DE GERÊNCIA DE BIG DATA

A seção anterior (2.2) apresenta detalhes sobre arquiteturas de sistemas de gerência de Big Data que realizam processamento paralelo através de um *cluster* de máquinas ou plataforma de nuvem. A maioria destes sistemas gerencia cada máquina como sendo um único tipo de nó (ABOUZEID et al., 2009; ALSUBAIEE et al., 2014; BITTORF et al., 2015; DAS et al., 2013; ISARD et al., 2007; MALEWICZ et al., 2010; MATTOSO et al., 2005; WANG, Jingjing et al., 2017). Da mesma forma, exigem que exista um disco de dados acoplado como ilustra a Figura 2.19. Estes sistemas exploram a tecnologia *multi-core* através do processamento *multi-thread*.

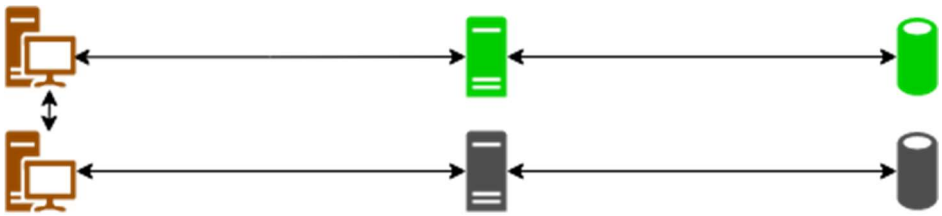

Figura 2.19: Arquiteturas com um nó por máquina e disco de dados acoplado

Alguns sistemas exploram o paralelismo intra-máquina atribuindo vários nós para uma mesma máquina. Nestas arquiteturas o recurso de armazenamento é compartilhado (DAGEVILLE et al., 2016) (Figura 2.20) ou fatiado (GUPTA et al., 2015; WARNEKE; KAO, 2009) (Figura 2.21).

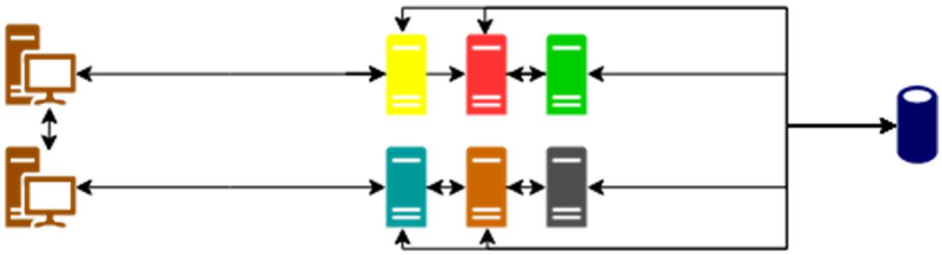

Figura 2.20: Arquitetura com vários nós por máquina e armazenamento compartilhado

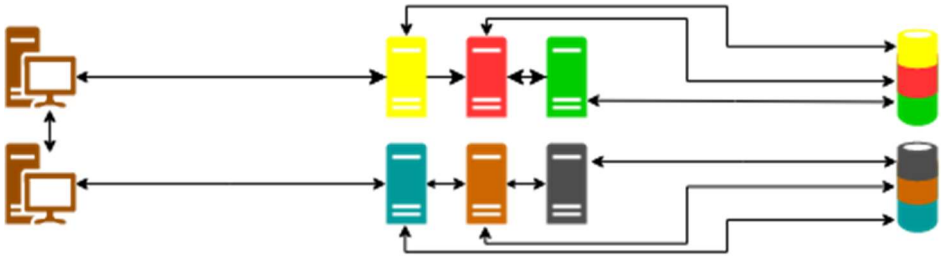

Figura 2.21: Arquitetura com vários nós por máquina e armazenamento fatiado

É fato que estas abordagens exploram a tecnologia *multi-core* através do processamento *multi-thread* ou atribuindo vários nós em uma mesma máquina. Contudo, nenhuma explora totalmente os recursos disponíveis de armazenamento e processamento de forma independente sem acesso concorrente. A Figura 2.22 ilustra uma arquitetura com nós independentes para processamento e armazenamento na mesma máquina. Nesta arquitetura, há vários nós que recebem operações onde não há necessidade de acesso a disco de dados (processamento). Por outro lado, cada controladora de disco de dados pertence a um nó que realiza operações de leitura e escrita de dados, bem como processamento, evitando acesso concorrente ao disco, e consequentemente, gargalos de I/O.

Figura 2.22: Arquitetura com vários nós independentes por máquina

Como mostra a arquitetura da Figura 2.22, várias atividades podem ser gerenciadas e processadas em paralelo por nós independentes, em uma mesma máquina, através do paralelismo intra-máquina, ou seja, enquanto um nó realiza operações de leitura no disco de dados, os demais nós iniciam o processamento destes dados (usando um esquema de *pipeline*) sem acesso concorrente ao disco.

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados detalhes sobre arquiteturas de sistemas de gerência de Big Data que realizam processamento paralelo através de um *cluster* de máquinas ou plataforma de nuvem. Além disso também foi avaliado o uso da tecnologia *multi-core* por estes sistemas. A Tabela 2.1 apresenta um resumo de como os Sistemas de Gerência de Big Data gerenciam máquinas em *cluster*.

Tabela 2.1: Gerenciamento de nós em *cluster* de Sistemas de Gerência de Big Data

Sistema	Data de Publicação	Nós por Máquina	Armazenamento	Tipo de Consulta
Hadoop	2004	Único	Acoplado	Java
HadoopDB	2009	Único	Acoplado	SQL
Apache Spark	2012	Único	Acoplado	Java, Scala, Python, R, Spark SQL, Mllib, GraphX e Spark Streaming
Asterix	2014	Único	Acoplado	AQL (*Asterix Query Language*)
Impala	2015	Único	Acoplado	SQL
ElasTraS	2013	Único	Acoplado	Query
Pregel	2010	Único	Acoplado	C++
Dryad	2007	Único	Acoplado	SQL e C++
Vertica	2012	Único	Acoplado	SQL, C++, Java, Python e R
Presto	2009	Único	Acoplado	SQL
ParGRES	2005	Único	Acoplado	SQL
Snowflake	2016	Múltiplos	Compartilhado	SQL
Nephele	2009	Múltiplos	Fatiado	DAG
Amazon Redshift	2015	Múltiplos	Fatiado	SQL
GreenPlum	2017	Múltiplos	Fatiado	SQL
MyriaX	2017	Único	Acoplado	JSON

A maioria destes sistemas gerencia cada máquina como *data nodes*, com recurso de processamento e armazenamento e exploram a tecnologia *multi-core* através do processamento *multi-thread*. Outros sistemas atribuem vários nós para uma mesma máquina, onde o recurso de armazenamento é fatiado ou compartilhado. Contudo, dentre todas as abordagens apresentadas, nenhuma explora totalmente os recursos disponíveis de armazenamento (discos de dados) e processamento (núcleos de CPU) de forma independente sem acesso concorrente, onde várias atividades podem ser gerenciadas e processadas em paralelo, por nós independentes, em uma mesma máquina através do paralelismo intra-máquina.

Para investigar o comportamento da abordagem apresentada na Figura 2.22 no processamento de consultas em sistemas de gerência de Big Data, escolhemos como base para os experimentos o mecanismo *shared-nothing* de execução de consulta MyriaX. Esta escolha foi motivada por este sistema permitir a alocação de nós independentes em uma mesma máquina

e também pelo co-orientador Professor Victor Almeida ter participado do desenvolvimento deste sistema em seu pós-doutorado. Além disso, acreditamos que a arquitetura *shared-nothing* nos permite uma melhor escalabilidade (DEWITT; GRAY, 1992; STONEBRAKER, 1986). A descrição detalhada do MyriaX é apresentada no Capítulo 3.

CAPÍTULO 3 – MYRIAX

3.1 INTRODUÇÃO

Este capítulo apresenta a descrição detalhada da arquitetura do MyriaX, um mecanismo de execução de consultas relacional *shared-nothing* e paralelo do sistema de gerência de big data Myria (WANG, Jingjing et al., 2017), bem como a descrição da abordagem de uso de nós. Também são apresentados detalhes do processamento paralelo *pipelined* de uma consulta de auto-junção (*self-join*). Selecionamos este mecanismo como base para nossos experimentos pois, além de utilizar uma arquitetura que permite gerenciar cada processador de uma máquina do *cluster* como um nó, o Prof. Dr. Victor Almeida, co-orientador desta pesquisa, trabalhou com o desenvolvimento deste sistema, o que facilitou o acesso ao código-fonte.

O restante deste capítulo apresenta a arquitetura do MyriaX (Seção 3.2), processamento paralelo de consultas (Seção 3.3) e considerações finais (Seção 3.4).

3.2 ARQUITETURA

A arquitetura do MyriaX é composta por um *master node* e *slave nodes*, seguindo a arquitetura *master-slave* (HALPERIN et al., 2014). O *master node* realiza o controle dos demais nós. Os *slave nodes* realizam o processamento dos dados e podem atuar como *data nodes* e/ou *worker nodes.*

Inicialmente, um nó é implantado como *data node.* A diferenciação entre *data node* e *worker node* acontece na atribuição de tarefas para cada nó. Nós que recebem operações de leitura e/ou escrita de dados atuam como *data nodes*. Por outro lado, nós que não recebem tarefas de leitura e/ou escrita de dados atuam como *worker nodes*. *Data nodes* podem também atuar como *worker nodes* após finalizar as operações de leitura e/ou escrita de dados. Desse modo, o MyriaX viabiliza a alocação de *data nodes* e *worker nodes* em uma mesma máquina, apesar disso nunca ter sido explorado pelo Myria. A Figura 2.18 ilustra uma implantação deste mecanismo com nove máquinas, sendo oito *data nodes*, além do *master node*.

Nesta figura, cada notação de cor laranja – *machine* – representa uma máquina física do *cluster*. Todas as máquinas contam com recursos de processamento – *CPU* – (notação azul mais escura) e armazenamento – *storage* – (notação azul mais clara). O fluxo de informações é representado por linhas, onde a linha tracejada representa o fluxo de informações de controle e a linha contínua representa fluxo de dados.

3.3 PROCESSAMENTO PARALELO DE CONSULTAS

Nesta seção são apresentados detalhes da metodologia de processamento paralelo utilizada pelo MyriaX. Inicialmente, é necessário que a base de dados tenha sido particionada no sistema. Para efeitos dessa dissertação, consideramos que os dados estão fragmentados e distribuídos entre vários nós utilizando a técnica *round-robin* (MEHTA; DEWITT, 1997). Esta estratégia de particionamento é adotada como padrão pelo MyriaX quando não é especificada nenhuma outra.

A técnica de particionamento *round-robin* entrega uma tupla da relação a ser particionada para cada nó designado, uma de cada vez. Os nós são dispostos em uma fila circular. A primeira tupla é enviada para o primeiro nó da fila, a segunda para o segundo nó, e assim por diante. Este processo se repete até que toda a relação tenha sido particionada. A Figura 3.1 ilustra este processo para dois nós.

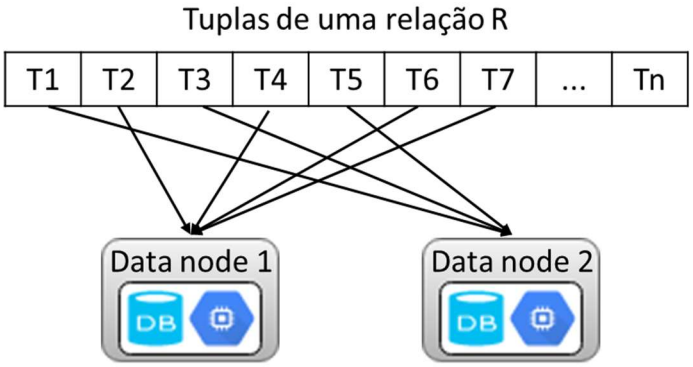

Figura 3.1: Particionamento *round-robin*

Para exemplificar o processamento paralelo, utilizamos uma consulta de auto-junção (*self-join*) (MISHRA; EICH, 1992). Neste tipo de consulta, as duas entradas para o operador de junção remetem à mesma fonte de dados, ou seja, é realizada junção da mesma tabela de dados. A operação de junção pode ser realizada sob diversos algoritmos de junção, tais como *nested-loop-join*, *sort-merge-join* e *hash-join* (SCHNEIDER; DEWITT, 1989). Neste exemplo, consideramos o algoritmo de junção *hash-join*.

O algoritmo de junção *hash-join* envolve a construção de uma tabela *hash* para cada uma das duas relações de entrada (R e S, por exemplo). Nesta tabela, cada *bucket* (fatia) recebe um conjunto de chaves da tabela de dados em questão, calculado a partir de uma função *hash* (GARCIA-MOLINA et al., 2008). Esta estratégia consiste em duas fases: *build* e *probe*. Primeiramente, é calculada toda a entrada da fase *build* e, então, constrói-se a tabela *hash* na memória. Cada tupla é inserida em um *hash bucket* dependendo do valor *hash* calculado para sua chave. Na fase seguinte (*probe*), é calculada uma tupla de cada vez, e para cada tupla *probe*,

o valor *hash* da chave é calculado, o *hash bucket* correspondente é verificado e as combinações são produzidas.

A Figura 3.2 ilustra a fase *build*. Nesta imagem a função *hash* é aplicada sobre a relação *R* (conjunto de chaves) e, de acordo com o resultado, cada chave é direcionada ao *bucket* correspondente. O mesmo é feito para as tuplas da relação S (utilizando *buckets* diferentes dos usados para as tuplas de *R*). A quantidade de *buckets* utilizada nesta fase é, geralmente, igual à quantidade de *worker nodes* (nós de processamento) disponíveis. A função *hash* se baseia nesta quantidade de nós e na chave de entrada para calcular o *bucket* correspondente e direcionar a tupla pela rede. Cada *bucket* está sendo preenchido localmente em memória principal em cada *worker node*. Este processo corresponde à etapa *Producer* no MyriaX e é efetuada em *data nodes*.

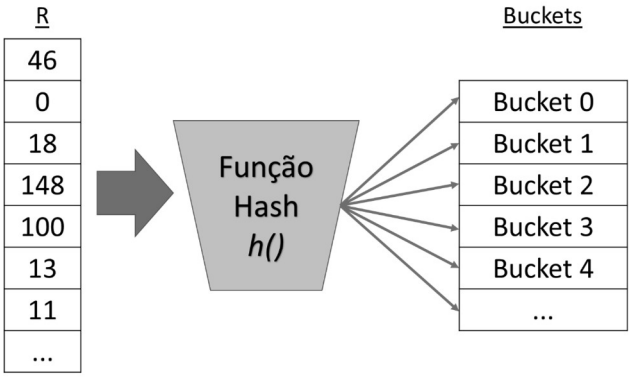

Figura 3.2: Fase *build* do algoritmo de *hash-join*

Na fase seguinte, a fase *probe*, os *buckets* das duas relações (R e S) estão em memória principal em cada *worker node* correspondente, dentro do conjunto de nós designados e de acordo com a função *hash* utilizada, para o processamento da junção $R \bowtie S$. No MyriaX este processo corresponde à etapa de *Consumer* e utilizada apenas o recurso de processamento dos nós. Ao fim do processamento da junção, cada *worker node* encaminha seus resultados para o *master node* para composição do resultado final. A Figura 3.3 ilustra este processo utilizando dois *data nodes* (*Producers*) e 4 *worker nodes* (*Consumers*).

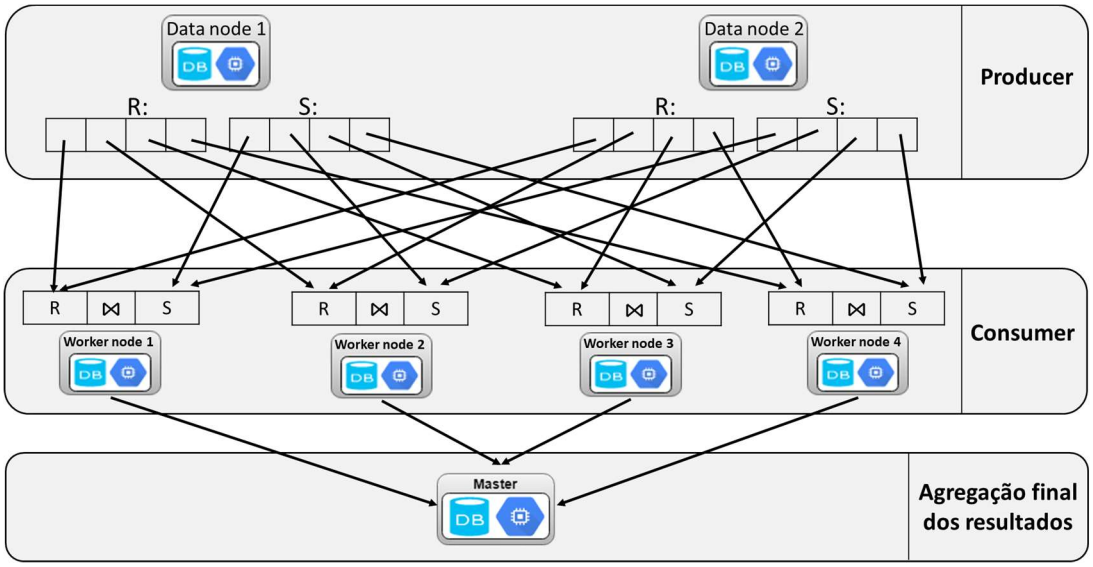

Figura 3.3: Fase *probe* do algoritmo de *hash-join*

No MyriaX, a função *hash* e as relações para o processamento de uma consulta são descritas em um plano paralelo de consulta (WANG, Jingjing et al., 2017). Este plano é composto por um conjunto de fragmentos e é passado ao MyriaX em formato JSON. Cada fragmento representa um conjunto de operações que são realizadas em um mesmo nó. A Figura 3.4 apresenta o plano de processamento paralelo da consulta de auto-junção.

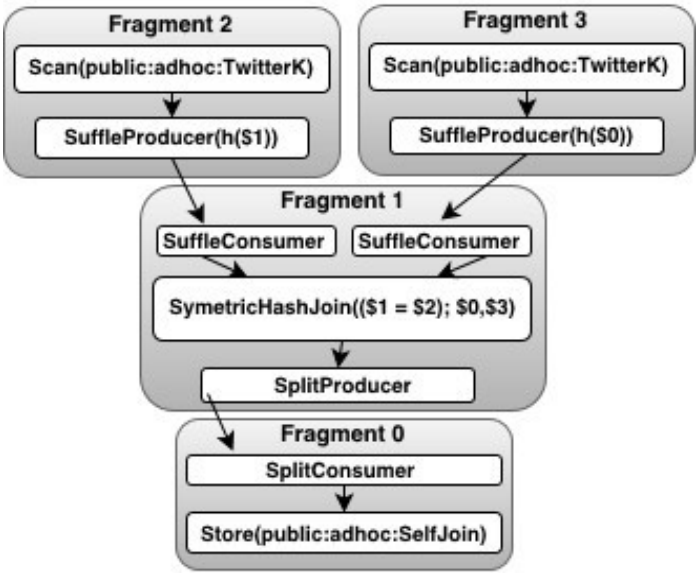

Figura 3.4: Plano de consulta de auto-junção

Neste plano, os Fragmentos 2 e 3 realizam a operação de leitura dos dados *Scan(public:adhoc:TwitterK)* (executados por *data nodes*), calculam o valor *hash* do atributo de junção e os encaminham para os nós da rede que irão efetuar a junção com as operações *SuffleProducer(h($1))* e *SuffleProducer(h($0))*, respectivamente. A operação de *Scan()* faz a

leitura da tabela *public:adhoc:TwitterK*. O Fragmento 1 (executado por *worker nodes* e/ou *data nodes* livres) recebe os dados enviados pelos Fragmentos 2 e 3 pela rede, preenchendo um *buffer* e disparando a execução da junção, utilizando o algoritmo de *hash-join* descrito anteriormente (SCHNEIDER; DEWITT, 1989), assim que este *buffer* fica cheio, de uma maneira não bloqueante. Os resultados obtidos são enviados para e salvos no Fragmento 0.

O MyriaX permite que se especifique quais nós cada fragmento irá utilizar para realizar suas operações. Aqueles que realizam operação de leitura de dados devem ser direcionados exclusivamente para *data nodes*. Aqueles que realizam processamento de operadores de álgebra relacional, por sua vez, podem ser direcionados para *worker nodes* e/ou *data nodes*.

O plano de consulta é gerado e passado ao MyriaX através de algum otimizador de consultas, tal como o RACO (WANG, Jingjing et al., 2017). A execução acontece em paralelo, com múltiplos fragmentos independentes ou encadeados (*pipelined*) entre nós, no caso comum do paralelismo de dados na avaliação da consulta. A partir da descrição do plano da consulta (em formato JSON), o MyriaX atribui cada fragmento ao seu conjunto de nós, tal como foram especificados, para processamento dos operadores. Cada fragmento inicia seu processamento a partir do momento que o anterior conclui parte de sua tarefa, em um *pipeline* de operações de álgebra relacional. Caso não haja fragmentos anteriores, como no caso de operações de leitura de dados, o *master node* inicia a execução do fragmento imediatamente.

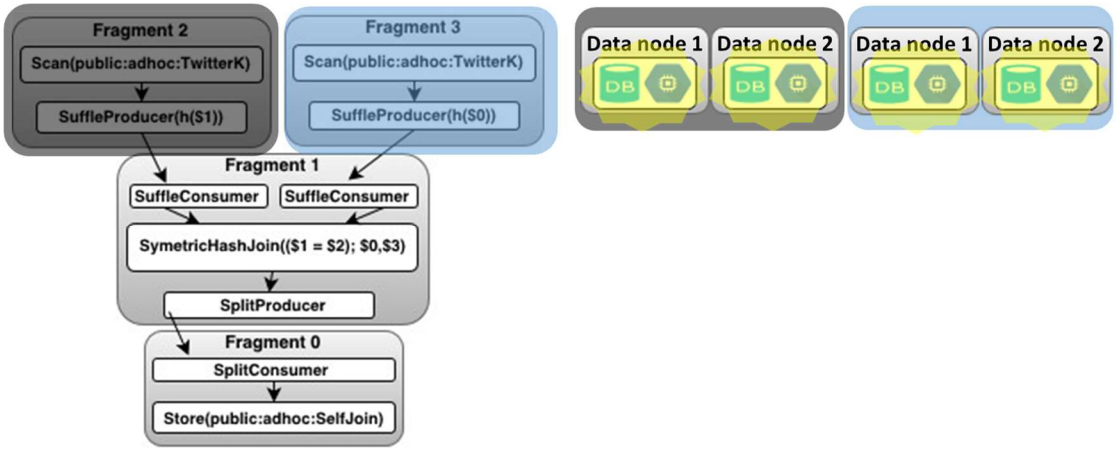

Figura 3.5: *Pipeline* de processamento do MyriaX (leitura de dados)

A partir do plano de consulta de auto-junção apresentado na Figura 3.4, a Figura 3.5 apresenta o início do processamento paralelo e *pipelined* deste plano. Neste exemplo, a base de dados está particionada em dois *data nodes*. Desta forma, os fragmentos de leitura de dados (2 e 3) são atribuídos a tais *data nodes*. Ambos iniciam imediatamente as operações de leitura de dados (*Scan()*) e utilizam tanto o recurso de processamento quanto de armazenamento, como destacado em amarelo na figura.

Após conclusão de parte da operação de leitura de dados, os resultados são encaminhados (de acordo com a chave *hash* calculada) para os quatro *worker nodes* para início do processamento do Fragmento 1. Este fragmento realiza a junção dos dados recebidos dos Fragmentos 2 e 3. Para tal, é utilizado apenas o recurso de processamento e todos os nós vinculados a este fragmento atuam como *worker nodes*. Cabe notar que nós que atuaram como *data nodes* podem ou não atuar como *worker nodes*, de acordo com a configuração estabelecida no MyriaX. A Figura 3.6 ilustra este processo e também destaca o recurso em uso pelos nós em amarelo.

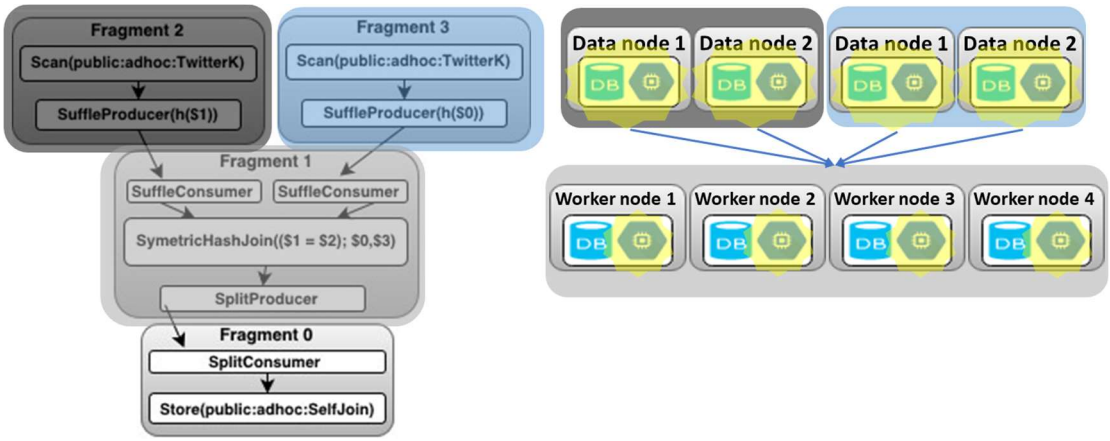

Figura 3.6: *Pipeline* de processamento do MyriaX (junção)

Por fim, os resultados da junção processada no Fragmento 1 são encaminhados ao Fragmento 0 para composição do resultado final. Este fragmento é de responsabilidade do *master node*, como apresentado na Figura 3.7.

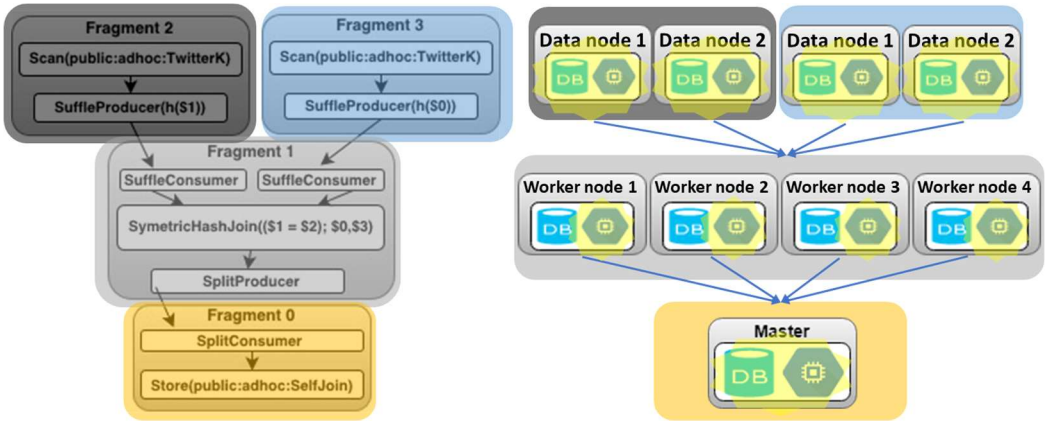

Figura 3.7: *Pipeline* de processamento do MyriaX (composição resultado final)

3.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a descrição da arquitetura do mecanismo de execução de consultas MyriaX. Além disto, também foi descrito o processamento paralelo utilizando o plano

de uma consulta de auto-junção. Com base nesta arquitetura, planejamos e executamos um conjunto de experimentos envolvendo alocações de *data nodes* e *worker nodes* (Capítulo 4) e processamento de cargas de trabalho (Capítulo 5).

CAPÍTULO 4 – EXPERIMENTO I

4.1 INTRODUÇÃO

Este capítulo apresenta o planejamento, execução e avaliação do Experimento I. Este experimento tem por finalidade confirmar a hipótese de que "*explorar todos os recursos de núcleos de processamento e discos de dados, de forma independente em cada máquina no cluster, diminui o tempo de processamento paralelo de consultas*". Tal hipótese está atrelada ao objetivo principal desta dissertação que é melhorar a eficiência do processamento paralelo de consultas em Big Data explorando recursos de armazenamento e processamento subutilizados disponíveis em cada máquina.

Neste contexto, este experimento buscou explorar todos os recursos disponibilizados por máquina do *cluster* como, por exemplo, núcleos de processadores com e sem discos de dados acoplados, para melhorar a eficiência no processamento da consulta. Para tal, foi utilizada a abordagem de *worker nodes* (nós que processam dados e não armazenam dados), e *data nodes* (nós que processam e armazenam dados) alocados em uma mesma máquina. Assim, operações que necessitam realizar leitura ou escrita no disco de dados são realizadas por *data nodes*. Por outro lado, operações que não necessitam de acesso a disco de dados, mas apenas processamento de dados, são realizadas por *worker nodes*.

A plataforma base da execução deste experimento foi o mecanismo de execução de consultas *shared-nothing* e paralelo MyriaX (WANG, Jingjing et al., 2017), descrito no Capítulo 3. Neste mecanismo, cada máquina que participa do processamento paralelo de consultas atua apenas como *data node*. Contudo, as tecnologias atuais permitem processamento paralelo em processadores que contêm vários núcleos. Em teoria, isto permite que uma mesma máquina tenha mais de um tipo de nó associado à um mesmo processador. Por exemplo, uma máquina que tenha um processador capaz de processar oito tarefas em paralelo (oito núcleos físicos) poderia ser utilizada com um *data node* e sete *worker nodes*. O MyriaX permite que isso seja feito, apesar de nunca ter explorado esse tipo de configuração.

O MyriaX não garante afinidade entre processos e núcleos do processador. Inicialmente, isso poderia ser um problema. No entanto, durante o desenvolvimento do código do MyriaX, a equipe de desenvolvimento realizou testes sobre afinidade entre processos e os núcleos do processador. Os resultados mostraram que forçar esta afinidade piorou o desempenho do sistema, e que deixar o sistema operacional escalonar tais processos resulta em melhor

desempenho. Dessa forma, é tarefa do sistema operacional garantir a afinidade entre processos e núcleos de processamento.

É importante notar que também é possível conseguir paralelismo de I/O, desde de que haja mais de uma controladora de disco por máquina. Isto porque os dados são particionados entre os vários *data nodes* e cada controladora pode remeter a um *data node*.

Para avaliar diversos cenários da abordagem proposta neste capítulo junto ao MyriaX, foram utilizadas duas consultas caras: uma de auto-junção (C1) (MISHRA; EICH, 1992) e outra de identificação de triângulos (C2) (HU et al., 2013). Tais consultas operam sob uma base de dados da rede social Twitter[14], que conta com uma tabela com duas colunas e grande volume de dados.

Este experimento avalia (i) o impacto do uso de *worker nodes* sem paralelismo de dados, ou seja, uso de uma máquina (um disco de dados); (ii) o impacto da memória *cache* no processamento das consultas; (iii) uso de *data nodes* atuando como *worker nodes*; e (iv) o processamento das consultas C1 e C2 em cenários com variações de *data nodes* e *worker nodes*. Os resultados destes experimentos foram descritos e avaliados em um artigo publicado no 32º Simpósio Brasileiro de Banco de Dados (SILVA et al., 2017).

No restante do capítulo é apresentada a Visão Geral do Experimento (Seção 4.2), Base de dados e Consultas (Seção 4.3), Planejamento (Seção 4.4), Resultados e Avaliação (Seção 4.5) e, por fim, as Considerações Finais (Seção 4.6).

4.2 VISÃO GERAL DO EXPERIMENTO

A abordagem proposta para os experimentos é de alocar vários *data nodes* e *worker nodes* em uma mesma máquina do cluster. A finalidade desta abordagem é explorar eficientemente os recursos disponíveis em cada máquina do cluster, tais como disco de dados e núcleos de processamento, e fornecer maior eficiência no processamento paralelo de consultas.

A alocação de nós depende do recurso disponível em cada máquina. A quantidade de *data nodes* está sujeita ao número de controladoras de disco de dados para evitar concorrência em operações de leitura de dados. A quantidade de *worker nodes* está sujeita ao número de núcleos de processamento disponíveis para evitar concorrência aos núcleos.

A Figura 4.1 ilustra esta abordagem. Nesta ilustração, há 3 máquinas em que uma é utilizada como *master node*, enquanto as demais contêm 3 *worker nodes* e 1 *data node* cada.

[14] https://dew-uff.github.io/myria-uff/

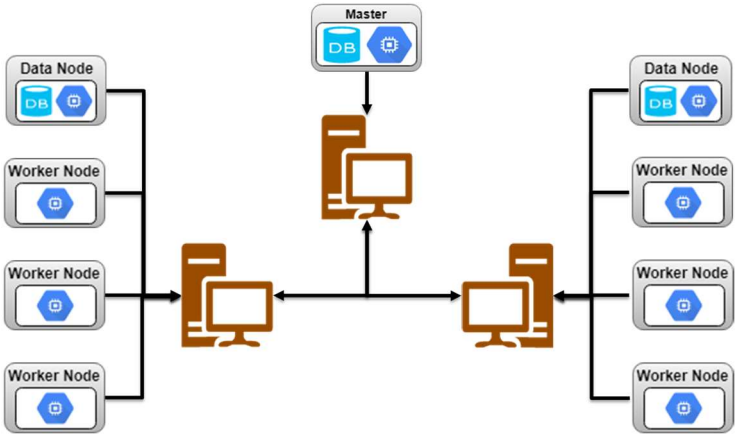

Figura 4.1: Alocações de *worker nodes* e *data nodes*

Alocar diversos nós em uma mesma máquina implica em paralelismo intra-máquina, pois nós que recebem dados de nós alocados na mesma máquina não usam o recurso de rede, evitando custos de transmissão e comunicação entre máquinas. Sendo assim, aplicamos esta abordagem no processamento paralelo de consultas nos experimentos descritos a seguir.

4.3 BASE DE DADOS E CONSULTAS

A base de dados utilizada para este experimento foi a do Twitter[15], que contém uma tabela com duas colunas (*follower* e *followee*) com valores de identificadores de usuários e representa relações entre usuários seguidores (*follower*) e seguidos (*followee*). Esta base contém 4.532.185 de tuplas e foi escolhida por ser uma base de dados real com grande volume de dados.

Para este experimento, foram utilizadas duas consultas caras referentes à base de dados do Twitter. A primeira consulta (C1) realiza auto-junção (*self-join*) (MISHRA; EICH, 1992) entre as colunas citadas da base, com a finalidade de identificar relações entre usuários (por exemplo identifica que o usuário A segue B, e que o usuário B segue C). O plano desta consulta para o MyriaX é apresentado e descrito no Capítulo 3.

A segunda consulta (C2) identifica triângulos entre usuários. Um triângulo se forma quando um usuário A segue um usuário B, que por sua vez segue um usuário C, que segue o usuário A. De fato, esta é uma consulta que ainda apresenta desafios para a comunidade de banco de dados quando os dados não cabem na memória (HU et al., 2013). A Figura 4.2 apresenta um grafo não direcionado ilustrativo G considerando a base de dados em questão, onde os vértices são representados pelos identificadores dos usuários ($U1$, $U2$, ... , $U11$) e as arestas representam a relação entre seguido e seguidor. Os triângulos, relações formadas por três vértices, estão em destaque.

[15] https://dew-uff.github.io/myria-uff/

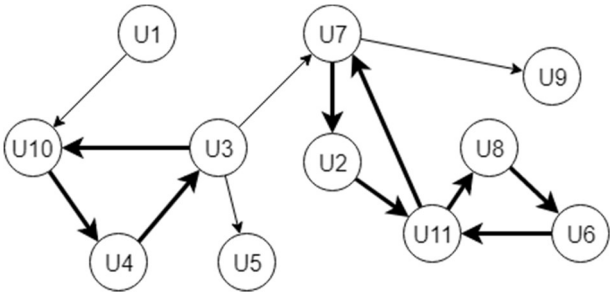

Figura 4.2: Representação de um triângulo (subgrafo) em um grafo *G*

O plano da consulta C2 é apresentado na Figura 4.3. Neste plano, a primeira junção é realizada no Fragmento 2 a partir da leitura produzida pelos Fragmentos 4 e 5. O resultado do Fragmento 2 é então passado ao Fragmento 1, que também recebe o resultado da leitura do Fragmento 3. O Fragmento 1 realiza uma nova junção com as duas entradas e encaminha o resultado ao Fragmento 0, que salva estes resultados. Na base de dados utilizada no experimento, a primeira e segunda junções (Fragmentos 2 e 1) retornam 2.045.216.395 e 89.084.893 de tuplas, respectivamente.

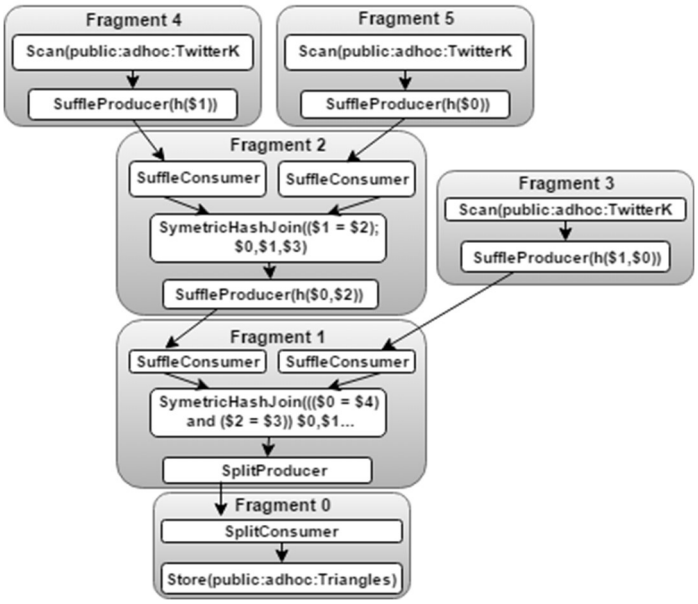

Figura 4.3: Plano de consulta de contagem de triângulos

4.4 PLANEJAMENTO

Por padrão, o MyriaX utiliza cada máquina como sendo um único tipo de nó (*data node*). Neste caso, para alocar mais de um tipo de nó em cada máquina utiliza-se portas de rede diferentes. Tais portas são descritas no arquivo de configuração utilizado para a implantação do serviço. Em cada máquina a quantidade de *data nodes* alocada foi igual à quantidade de discos

de dados disponíveis e a quantidade de *worker nodes* foi igual à quantidade de núcleos de processamento disponíveis, excluindo-se os núcleos alocados para *data nodes*.

O *cluster* (Oscar) utilizado para execução dos experimentos é de responsabilidade do Instituto de Computação da Universidade Federal Fluminense e, portanto, está disponível para a comunidade acadêmica. Este *cluster* é composto de 42 máquinas, cada uma com 2 processadores Intel Xeon *quadcore* (8 núcleos), 16 GB memória de RAM e 160 GB de disco. Todas as máquinas são interligadas com *switch Gigabit Ethernet*. O sistema operacional é o *Red Hat Enterprise Linux Server* versão 5.3.

Neste experimento, utilizamos um *script* que, a partir de uma quantidade de *worker nodes* e *data nodes*, gera vários cenários de distribuição e alocação de *data nodes* e *worker nodes* em máquinas do *cluster*. Este *script* realiza 5 rodadas de consultas para cada cenário gerado. Cada rodada de consultas é executada para todos os cenários gerados antes de iniciar a rodada seguinte, ao invés de todas as 5 rodadas de consultas serem executadas de forma sequencial para cada cenário. Com esta lógica, evita-se que um cenário seja totalmente prejudicado por alguma atividade de manutenção do sistema operacional que fuja ao nosso controle durante a execução das consultas. Além disso, para avaliar o impacto de uso de memória *cache*, cada rodada é constituída de 2 execuções sequenciais da consulta. Os tempos das 5 primeiras execuções de cada consulta são usados para avaliação nos cenários sem memória *cache*. Já os tempos das 5 segundas execuções são usados para avaliar o impacto de memória *cache*. A Figura 4.4 apresenta o fluxograma deste *script*.

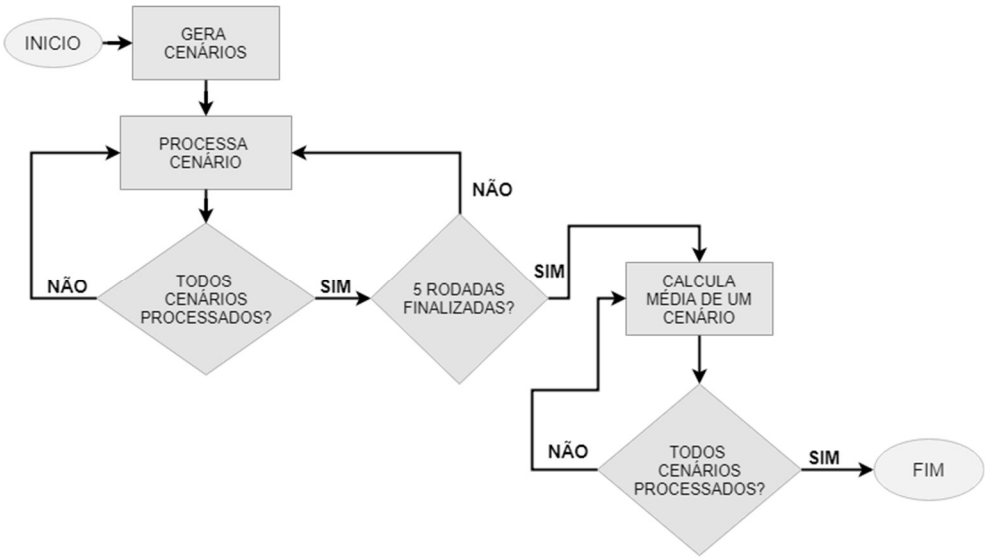

Figura 4.4: Fluxograma de execução do *script*

Cada cenário é composto por uma quantidade de máquinas (*m*), *data nodes* (*dn*) e *worker nodes* (*wn*), onde a quantidade de *worker nodes*, quando maior que zero, é dividida

igualmente para cada máquina do cenário. A Tabela 4.1 apresenta a distribuição de tais cenários. A heurística para criar os cenários se baseou em valores de potência de 2 (iniciando em 2) para a quantidade de nós em cada implantação do serviço Myria em *cluster*, tendo como limite a quantidade máxima de núcleos de processadores disponibilizados, que para esse experimento foi um processador com 8 núcleos. Em cada implantação o algoritmo determina a quantidade para cada tipo de nó. Para *data nodes*, também foram utilizados valores de potência de 2 (iniciando em 2), tendo como limite a quantidade de máquinas com discos de dados disponibilizados pelo *cluster*. Os demais nós são alocados como *worker nodes*, uma vez que há uma única controladora de disco por máquina.

Tabela 4.1: Cenários para 8 máquinas com 8 núcleos de processamento cada

Cenário	# máquinas	# *data nodes*	# *worker nodes*	# nós de processamento
m:2_dn:2_wn:0	2	2	0	2
m:2_dn:2_wn:2	2	2	2	4
m:2_dn:2_wn:6	2	2	6	8
m:2_dn:2_wn:14	2	2	14	16
m:4_dn:2_wn:30	4	2	30	32
m:8_dn:2_wn:62	8	2	62	64
m:4_dn:2_wn:2	4	2	2	4
m:4_dn:4_wn:0	4	4	0	4
m:4_dn:4_wn:4	4	4	4	8
m:4_dn:4_wn:12	4	4	12	16
m:4_dn:4_wn:28	4	4	28	32
m:8_dn:4_wn:60	8	4	60	64
m:8_dn:2_wn:6	8	2	6	8
m:8_dn:4_wn:4	8	4	4	8
m:8_dn:8_wn:0	8	8	0	8
m:8_dn:8_wn:8	8	8	8	16
m:8_dn:8_wn:24	8	8	24	32
m:8_dn:8_wn:56	8	8	56	64

A Tabela 4.1 apresenta os cenários possíveis para implantações em 9 máquinas, destacando os cenários padrão do MyriaX. Nesta tabela, a coluna "# nós de processamento" contabiliza a quantidade de nós atuando no *pipeline* de processamento (*data nodes* + *worker nodes*). Em todos os cenários, uma das máquinas é reservada para atuar como *master node* e, portanto, apenas as demais 8 máquinas são consideradas pelos cenários. Cabe ressaltar que os dados são particionados entre os *data nodes* usando a estratégia de *round-robin* (MEHTA; DEWITT, 1997).

A Figura 2.18 ilustra o possível cenário *m:8_dn:8_wn:0*. Este cenário oferece um maior particionamento da tabela de dados, pois utiliza 8 *data nodes*, armazenando uma menor quantidade de dados em cada nó de dados, em relação ao cenário *m:2_dn:2_wn:6*, ilustrado na Figura 4.5, que conta com 2 *data nodes* e 6 *worker nodes*, além do *master node*. Este último

cenário, por sua vez, sofre menor influência de tráfego de rede na transmissão de dados entre nós, pois contém 2 máquinas com 1 *data node* e 3 *worker nodes* cada, aumentando assim o paralelismo intra-máquina. Além disso, *data nodes* encaminham parte dos resultados de suas operações durante as mesmas para os nós seguintes da *pipeline* de processamento, ou seja, *worker nodes* iniciam suas atividades antes dos *data nodes* finalizarem suas operações, evitando, assim, execução sequencial dos operadores. Ambos os cenários, após concluídas as operações de leitura de dados, terão 8 nós (2 *dn* + 6 *wn*) disponíveis para o processamento paralelo da consulta.

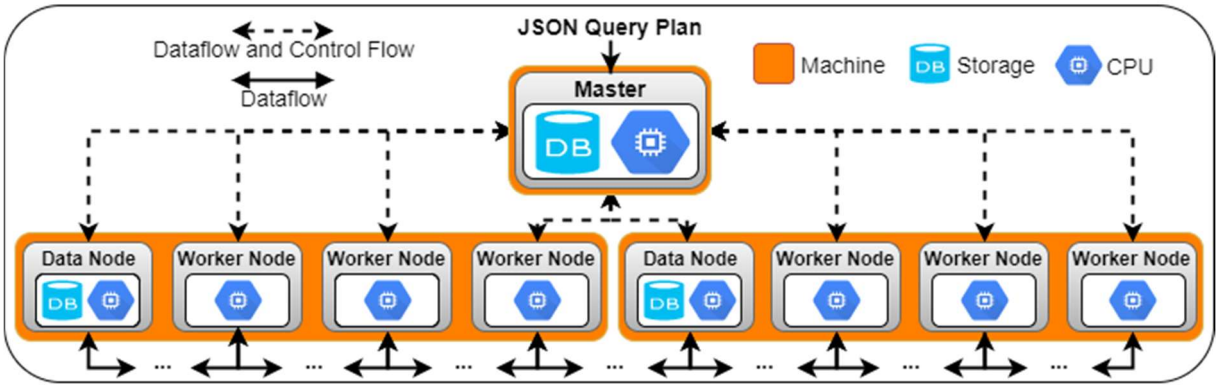

Figura 4.5: Implantação MyriaX com 3 máquinas (*m:2_dn:2_wn:6*)

Na primeira parte deste experimento, utilizamos apenas uma máquina para avaliar o comportamento da abordagem proposta sem particionamento da base de dados (cada máquina contém apenas uma controladora de disco de dados) com adição de *worker nodes* tendo como limite a quantidade de núcleos de processamento disponíveis. Sendo assim, a finalidade deste experimento é investigar o desempenho do processamento paralelo das consultas C1 e C2 com cenários que contam com 1 *data node* e até 7 *worker nodes*.

Na segunda parte deste experimento realizamos o processamento das consultas C1 e C2 sobre os cenários apresentados na Tabela 4.1 de duas formas: (a) *data nodes* realizando apenas operações de leitura e escrita de dados e (b) *data nodes*, além de realizar operações de leitura e escrita de dados, atuando no *pipeline* de processamento das operações de junção como *worker nodes*.

Durante as submissões das consultas, o *script* captura o tempo que cada consulta leva para executar e calcula a média, eliminando as consultas com maior e menor tempo, para cada cenário. Os resultados obtidos são apresentados e avaliados na próxima seção.

4.5 RESULTADOS E AVALIAÇÃO

Neste experimento realizamos quatro sub experimentos: comportamento da abordagem de *data nodes* e *worker nodes* com o paralelismo de processamento intra-máquina utilizando apenas uma máquina; impacto de memória cache no processamento das consultas C1 (auto-junção) e C2 (contagem de triângulos); resultados para consulta C1 e para consulta C2. Os resultados destes experimentos foram descritos e avaliados em artigo publicado no 32º Simpósio Brasileiro de Banco de Dados (SILVA et al., 2017).

Nos resultados experimentais apresentados nesta seção, cenários padrão do MyriaX, ou seja, aqueles utilizados por este sistema, estão nomeados como *Baseline* e cenários que se baseiam na hipótese apresentada nesta dissertação são nomeados *Avaliação*. O cálculo do percentual de aceleração/desaceleração em cenários *Avaliação* utilizou a seguinte fórmula com base em DeWitt e Gray (1992):

$$\left(\left(\frac{T1}{T2}\right) * 100\right) - 100$$

DeWitt e Gray (1992) não apresentam a fórmula de aceleração (*speed up*) em forma de percentual. Sendo assim, para transformar o resultado em percentual, acrescentamos uma multiplicação por 100 e posterior subtração de 100 à fórmula destes autores. O cálculo de aceleração em cenários *Avaliação* em relação ao cenário *Baseline* utiliza a constante T1 igual ao tempo gasto no cenário *Baseline* e a constante T2 igual ao tempo gasto no cenário *Avaliação*. Por outro lado, o cálculo de desaceleração em cenários *Avaliação* em relação ao cenário *Baseline* utiliza a constante T1 igual ao tempo gasto no cenário *Avaliação* e a constante T2 igual ao tempo gasto no cenário *Baseline*.

Além disso, também são apresentados e avaliados os resultados para experimentos com influência de memória cache. Todos os tempos discutidos nessa seção são medidos em segundos.

4.5.1 WORKER NODES SEM PARALELISMO DE DADOS

Para avaliar o impacto de *worker nodes* sem paralelismo de dados, realizamos um experimento com apenas 1 máquina (Figura 4.6) ou seja, sem particionamento da base de dados e tráfego de rede (*overhead*), utilizando no máximo 7 *worker nodes*, totalizando 8 nós de processamento, explorando todos os núcleos de processamento disponibilizados. O cenário *Baseline m:1_dn:1_ wn:0* apresentou o maior tempo médio para ambas as consultas. Isto porque

o único *data node* desse cenário processou todos os operadores das consultas de forma sequencial.

Os cenários *Avaliação* de ambas as consultas, por outro lado, utilizaram de paralelismo intra-máquina durante operações de junção devido ao uso de *worker nodes* e obtiveram menor tempo médio de processamento, com destaque para o cenário *m:1_ dn:1_wn:7* que explora todo o recurso de núcleos de processamento disponibilizados. O tempo nesse cenário implicou aceleração de 206,66% para a consulta C1 e 244,37% para a consulta C2, em relação ao cenário *Baseline m:1_dn:1_wn:0*, mesmo sem paralelismo de I/O.

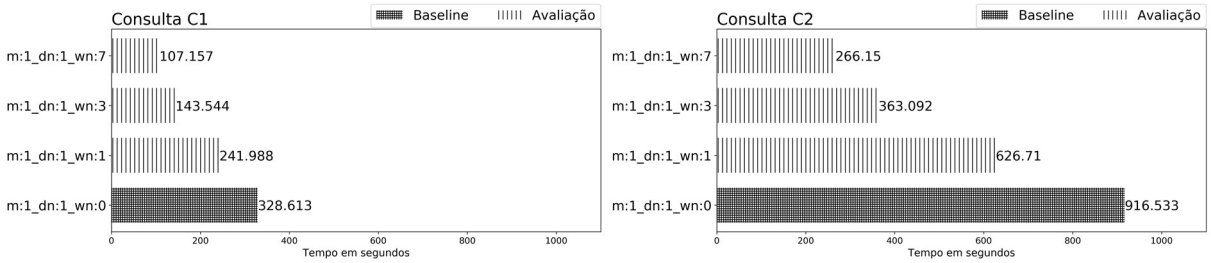

Figura 4.6: Tempo médio para C1 e C2 referente aos cenários de 1 máquina

4.5.2 USO DE DATA NODE COMO WORKER NODE

Neste experimento realizamos a execução das consultas C1 e C2 utilizando os cenários da abordagem proposta nesta dissertação. Foram realizadas duas execuções: (a) *data nodes* realizando apenas operações de leitura e escrita de dados (DN não atua como WN) e (b) *data nodes*, além de realizar operações de leitura e escrita de dados, atuando no *pipeline* de processamento das operações de junção como *worker nodes* (DN atua como WN). Os resultados da consulta C1 são apresentados na Figura 4.7 e da consulta C2 na Figura 4.8.

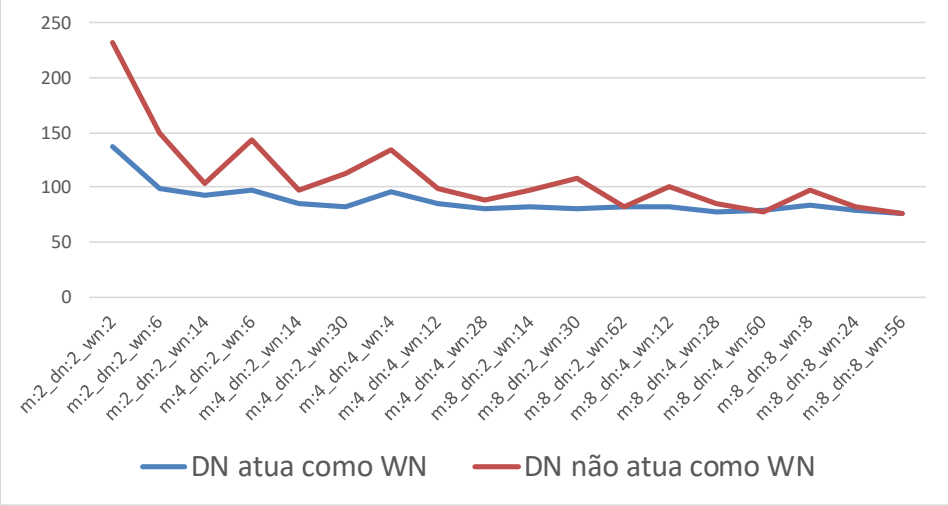

Figura 4.7: Tempo médio para C1 em experimento com uso de *data node*

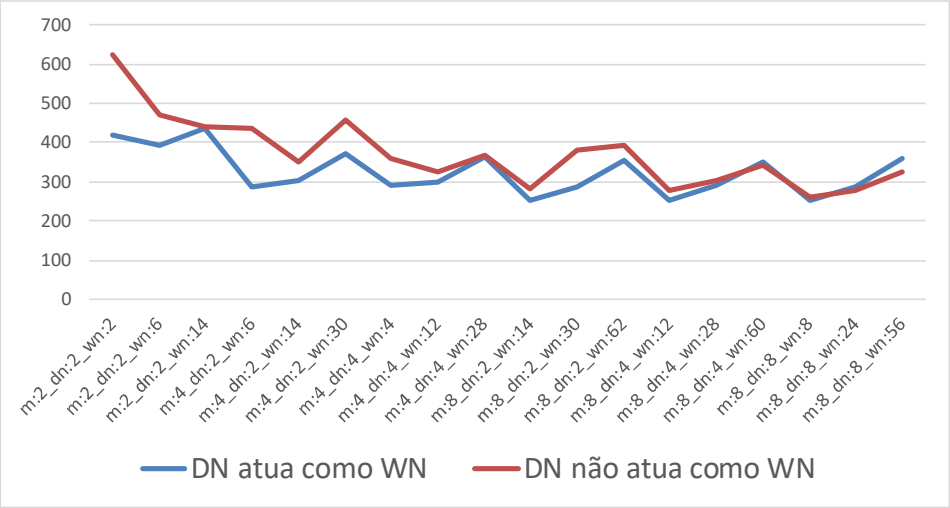

Figura 4.8: Tempo médio para C2 em experimento do uso de *data node*

Os resultados apresentados para este experimento mostram que cenários que utilizaram *data nodes* atuando também como *worker nodes*, ou seja, atuando no processamento de junções, tiveram melhor desempenho em relação a cenários que utilizaram *data nodes* apenas para realizar operações de leitura e escrita de dados. Desta forma, os demais experimentos seguem o método de execução que utiliza *data nodes* atuando também como *worker nodes*, além de realizar leitura e escrita de dados.

4.5.3 IMPACTO DE MEMÓRIA CACHE

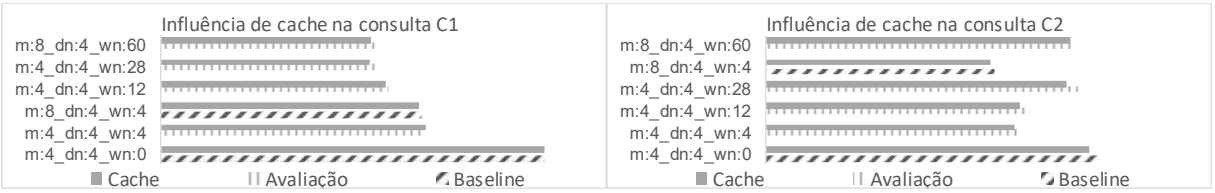

Figura 4.9: Tempo médio para C1 e C2 com influência de memória *cache*

A Figura 4.9 apresenta os resultados que incluem tempo médio das consultas C1 e C2 com influência de memória cache. Os cenários com maior influência de memória *cache* para a consulta C1 foram *m:4_dn:4_wn:28* e *m:8_dn:4_wn:60* com aceleração de aproximadamente 3,7%, e para a consulta C2 foi *m:4_dn:4_wn:28* com aceleração de aproximadamente 4,23%. Os demais cenários apresentam menor influência de memória *cache* no processamento da consulta. Os resultados mostram que a *cache* tem pouca influência no tempo de processamento das consultas. Esta pouca influência pode ser justificada pelo grande volume de dados destas consultas em que a memória *cache* não suportou armazenar todos os resultados, o que acarretou em *cache misses* recorrentes.

Por esse motivo, nos demais experimentos, usamos dados das execuções sem influência de memória *cache*.

4.5.4 CONSULTA C1 (AUTO-JUNÇÃO)

No processamento da consulta C1, o cenário *Baseline m:2_dn:2_wn:0* apresentou aceleração de aproximadamente 24,48% quando comparado ao cenário *Baseline m:1_dn:1_wn:0* da Figura 4.6, que utiliza uma máquina. Tal aceleração é justificada pelo particionamento e processamento da base de dados entre dois *data nodes* e apresentada na Figura 4.10.

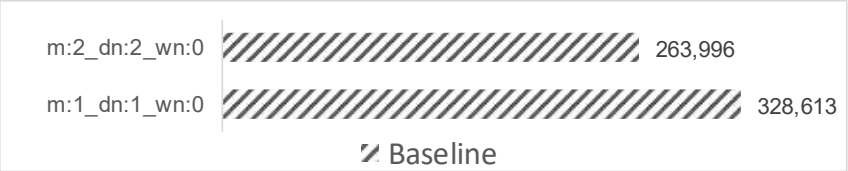

Figura 4.10: Cenários *Baseline* com 1 e 2 máquinas

A Figura 4.11 apresenta os resultados para a consulta C1 com uso de mais de um *data node*. Logo, há particionamento e paralelismo na leitura de dados. Para todas as quantidades de máquinas, os cenários *Avaliação* apresentam melhor desempenho em relação aos cenários *Baseline*. Em cenários com até 2 máquinas, o cenário *m:2_dn:2_wn:14* (melhor desempenho) apresentou aceleração de 192,47% em relação ao cenário *Baseline m:2_dn:2_wn:0*, que apresentou o maior tempo. Em cenários com até 4 máquinas, os cenários com melhores desempenho, *m:4_dn:4_wn:28 e m:4_dn:2_wn:30*, apresentaram aceleração de aproximadamente 77,67% e 75% respectivamente, em relação ao cenário *Baseline m:4_dn:4_wn:0*, que apresentou o maior tempo. Em cenários com até 8 máquinas, cenários *Avaliação* apresentaram aceleração de até 32,3% em relação ao cenário *Baseline m:8_dn:8_wn:0*, que apresentou o maior tempo. Tais cenários apresentaram resultado semelhante pela grande quantidade de *worker nodes*, que processam menores quantidades de dados. Os cenários *Baseline* apresentam resultado semelhante. Os cenários *Avaliação* apresentam maior quantidade de *worker nodes* para processamento do operador de junção e também maior paralelismo intra-máquina, ou seja, utilizam até 7 *worker nodes* que não são influenciados pelo tráfego de rede quando recebem dados a partir do *data node* alocado na mesma máquina. Neste sentido, é importante notar que, para cenários com até 4 máquinas, a simples adição de *worker nodes* implicou em aceleração de até 77,67% quando se explora todo o recurso de núcleos de processamento disponíveis.

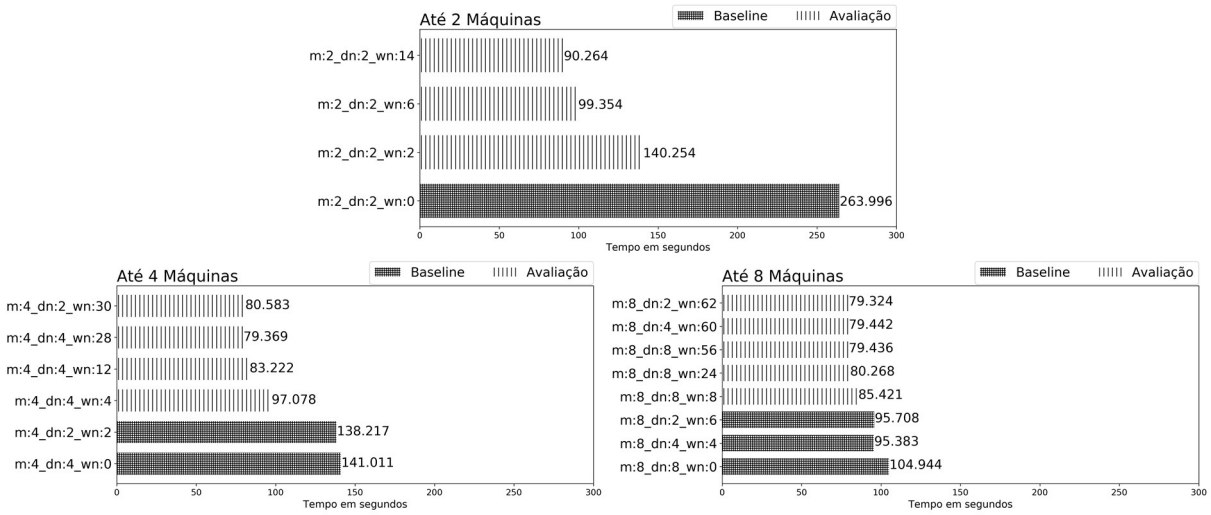

Figura 4.11: Tempo médio da consulta C1 com até 8 máquinas

Nos cenários com até 8 máquinas da Figura 4.11, os cenários *Avaliação* também apresentam melhor desempenho em relação aos *Baseline*, porém de forma menos acentuada. Os cenários *m:8_dn:2_wn:62*, *m:8_dn:4_wn:60* e *m:8_dn:8_wn:56*, que exploram todo o recurso de processamento e armazenamento disponível nas 8 máquinas, apresentam aceleração de aproximadamente 32,3% em relação ao cenário *m:8_dn:8_wn:0* que apresentou o maior tempo médio. Por conter apenas *data nodes*, este cenário teve seu desempenho influenciado pela execução sequencial dos operadores da consulta. Logo, nota-se que, para cenários com até 8 máquinas, a simples adição de *worker nodes* implicou em aceleração de até 32,3% quando se explora todo o recurso de núcleos de processamento disponíveis.

A Figura 4.12 apresenta os resultados para a consulta C1, utilizando cenários com até 8 máquinas agrupados pela quantidade total de nós (8, 16, 32 e 64). Cenários com 8 nós totais apresentaram o maior tempo médio para execução da consulta C1. O cenário que não utiliza *worker nodes* apresentou o maior tempo médio dentre estes cenários, justificado pela execução sequencial dos operadores da consulta. Os demais cenários com quantidade totais de 16, 32 e 64 nós apresentam melhora gradativa no tempo médio do processamento da consulta C1, respectivamente, e apresentam resultados semelhantes dentro destas quantidades de nós totais. Isto porque estes cenários utilizam de maior quantidade de *worker nodes* que passam a processar pequenas quantidades de dados e aumentam o tráfego de rede.

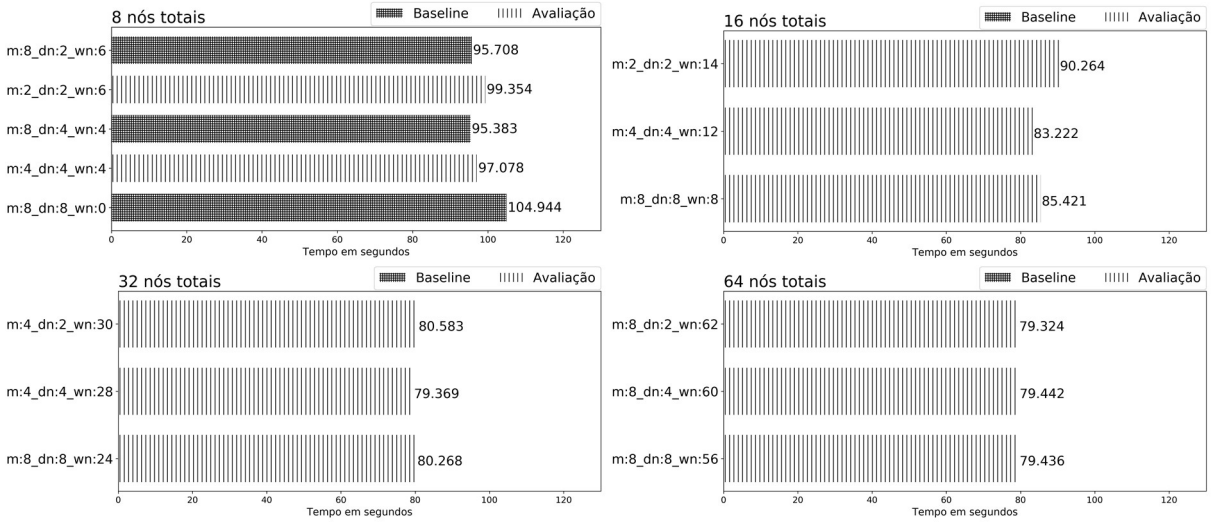

Figura 4.12: Tempo médio da consulta C1 em cenários com até 8 máquinas agrupados pela quantidade total de nós

A Figura 4.12 evidencia que, para a consulta C1, é possível atingir um determinado desempenho com menor quantidade de máquinas dimensionando *worker nodes* e *data nodes* e explorando recursos de núcleos de processamento e discos de dados de uma mesma máquina. Por exemplo, o cenário *m:8_dn:4_wn:4* utiliza 8 máquinas e apresentou desempenho inferior aos cenários *m:2_dn:2_wn:14* e *m:4_dn:4_wn:28* que utilizam 2 e 4 máquinas, respectivamente. Em ambientes onde há custo para alocar máquinas, isso significa uma economia importante de recursos.

4.5.5 CONSULTA C2 (CONTAGEM DE TRIÂNGULOS)

A Figura 4.13 apresenta o tempo médio de 5 rodadas da consulta C2.

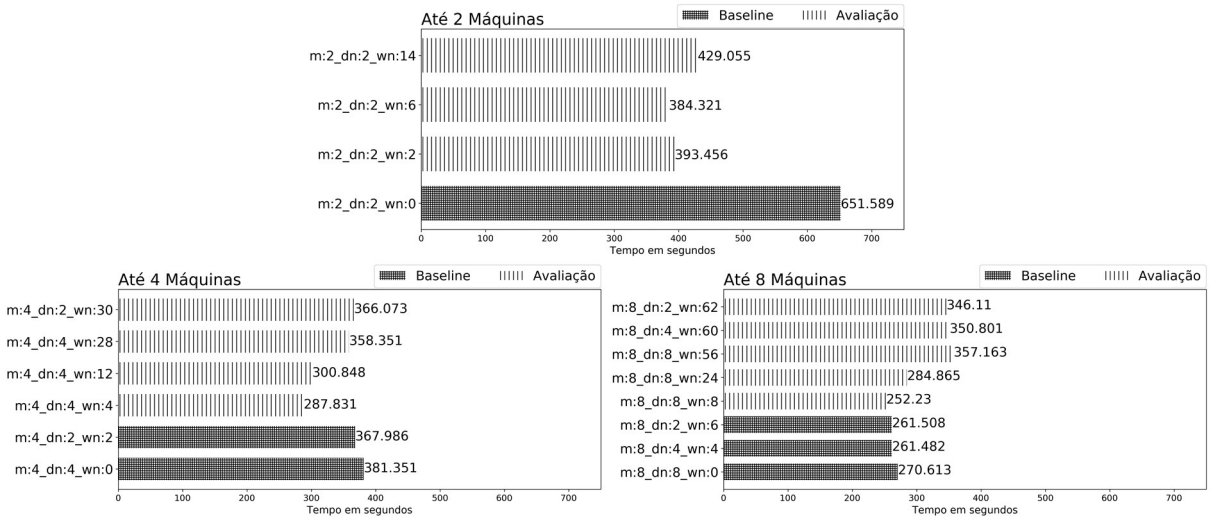

Figura 4.13: Tempo médio da consulta C2

Os cenários da Figura 4.13 com até 2 máquinas mostram que cenários *Avaliação* apresentaram menor tempo médio em relação ao cenário *Baseline*. O cenário *Avaliação m:2_dn:2_wn:6* apresentou aceleração de 69,54% em relação ao cenário *Baseline m:2_dn:2_wn:0*. Tal disparidade de tempos é influenciada pela execução sequencial dos operadores da consulta C2 realizada pelo cenário *Baseline*. Cabe notar que o aumento de *worker nodes* em cenários *Avaliação* com 2 máquinas apresentou aceleração de até 69,54% em relação ao cenário *m:2_dn:2_wn:6* e 51,87% em relação ao cenário *m:2_dn:2_wn:14,* ou seja, houve desaceleração no cenário com 14 *worker nodes* quando comparado ao cenário de 6 worker nodes. Isto ocorre porque, apesar de maior quantidade de *worker nodes* disponíveis para o processamento paralelo da consulta C2, há um maior tráfego de rede durante o fluxo dados para os *worker nodes* entre 2 máquinas. Este aumento no tráfego acontece pois a primeira auto-junção gera grande quantidade de possibilidades com dois vértices que são mantidas em memória enquanto a terceira operação de leitura de dados e a segunda junção são realizadas.

Nos cenários com até 4 máquinas da Figura 4.13, os cenários *Baseline* apresentam diferentes tempos médios devido à existência ou não de *worker nodes*. O cenário *m:4_dn:4_wn:0*, apesar de utilizar mais máquinas, apresenta maior tempo médio pela execução sequencial dos operadores, justificada pela ausência de *worker nodes*. O cenário *m:4_dn:2_wn:2*, por sua vez, tem melhor desempenho por ter *pipeline* no processamento da consulta com 2 *worker nodes*. O cenário *Avaliação* com melhor tempo médio foi o *m:4_dn:4_wn:4*, com aceleração de 27,85% em relação ao melhor cenário *Baseline* (*m:4_dn:2_wn:2*). Da mesma forma que na avaliação anterior, o aumento de *worker nodes* em uma dada quantidade de máquinas causa desaceleração no processamento da consulta C2. Tal desaceleração, causada por gargalos de interferência e comunicação, já era esperada, como discutido por Stonebraker (1986). Desta forma, o aumento de *worker nodes* causou uma desaceleração gradativa entre os cenários, chegando até aproximadamente 27,18% com os cenários *m:4_dn:4_wn:28* e *m:4_dn:2_wn:30*, que exploram todo o recurso disponível de 4 máquinas, em relação ao cenário *m:4_dn:4_wn:4*.

Nos cenários com até 8 máquinas da Figura 4.13, os cenários *Baseline* com uso de *worker nodes* também apresentam melhor desempenho em relação a cenários que contêm apenas *data nodes*, com aceleração de até 3,48%. O cenário *Avaliação m:8_dn:8_wn:8* apresentou aceleração de até 3,67% em relação ao cenário *Baseline* com melhor desempenho. Esse foi o cenário que apresentou o melhor desempenho para as 8 máquinas. É importante notar que o cenário *Avaliação m:8_dn:8_wn:56* apresentou maior tempo médio no processamento da consulta C2 com desaceleração de 46,74% em relação ao cenário *Baseline* com melhor

desempenho (*m:8_dn:4_wn:4*) e 52,12% em relação ao cenário *Avaliação* de melhor desempenho (*m:8_dn:8_wn:8*), para esta quantidade de máquinas. Isto porque, como dito anteriormente, há maior influência do tráfego de rede por explorar todos os núcleos de processamento com *worker nodes* em 8 máquinas (STONEBRAKER, 1986).

A Figura 4.14 apresenta os resultados para a consulta C2, utilizando cenários com até 8 máquinas agrupados pela quantidade total de nós (8, 16 e 32 e 64). O cenário *Avaliação m:8_dn:8_wn:8* apresentou ganho de aproximadamente 3,68% de tempo de processamento em relação ao cenário *m:8_dn:2_wn:6*, que apresentou menor tempo médio entre os *Baseline*.

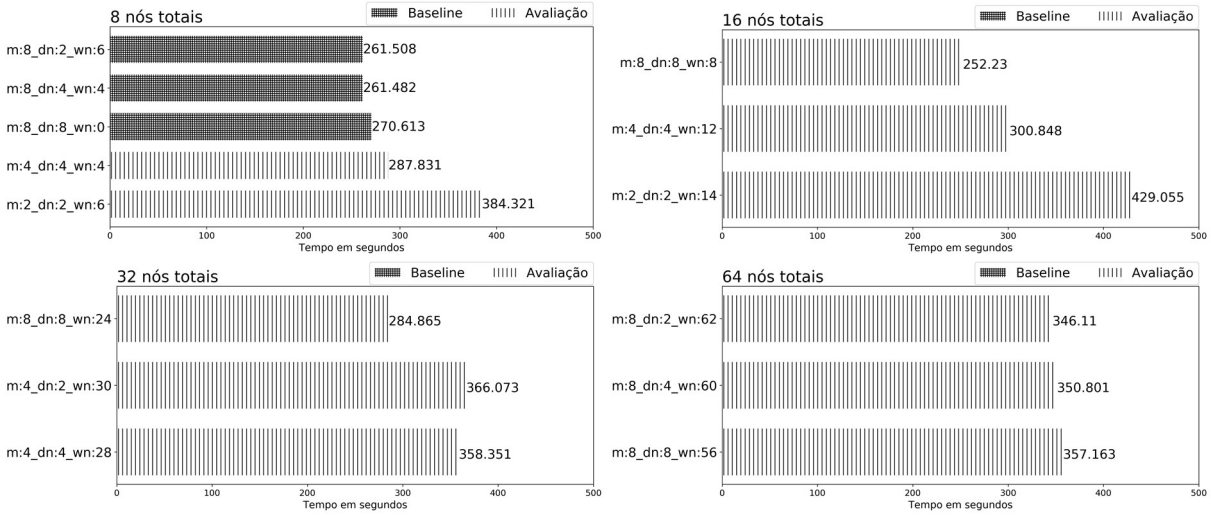

Figura 4.14: Tempo médio da consulta C2 referente aos cenários com até 8 máquinas agrupados pela quantidade total de nós

Cabe notar que para cenários de 8, 16 e 32 nós, que permitem variar quantidade de máquinas, o tempo médio de processamento da consulta C2 diminui com o aumento da quantidade de máquinas, ou seja, do paralelismo de I/O. No entanto, explorar totalmente o recurso de núcleos de processamento disponíveis para dada quantidade de máquinas causa um aumento no tempo de processamento devido ao gargalo formado pelo tráfego de rede (STONEBRAKER, 1986). Neste sentido, o cenário *Baseline m:8_dn:2_wn:6* apresentou aceleração 47% sobre o cenário *Avaliação m:2_dn:2_wn:6*, sendo que ambos usam a mesma quantidade de *worker nodes* e *data nodes*. Isto porque o primeiro usa 8 máquinas (menor gargalo de rede), enquanto o segundo conta com 2 máquinas. O mesmo acontece para cenários com 16 e 32 nós totais. Cenários com 64 nós totais apresentam resultados semelhantes, pois usam a totalidade dos recursos disponíveis em 8 máquinas, variando apenas a quantidade de data nodes. A Figura 4.14 mostra que, para a consulta C2, o uso de todo o recurso disponível em dada quantidade máquinas piora o desempenho de processamento. Por outro lado, cenários *Avaliação* com quantidades iguais de *data nodes* e *worker nodes* (*m:8_dn:8_wn:8*,

m:4_dn:4_wn:4 e *m:2_dn:2_wn:2*), que utilizam 2 nós por máquina, apresentaram os melhores desempenhos para a consulta C2 com aceleração de até aproximadamente 70%.

4.6 CONSIDERAÇÕES FINAIS

Este capítulo avaliou o uso de *worker nodes*, que participam do *pipeline* de processamento da consulta e não armazenam dados, e *data nodes*, que armazenam dados e também atuam como *worker nodes*, alocados em uma mesma máquina. O mecanismo de execução de consulta *shared-nothing* MyriaX foi utilizado como plataforma base dos experimentos por permitir o uso desta abordagem.

Visando confirmar a hipótese de que é possível diminuir o tempo de processamento de consultas através da adição de *worker nodes* em núcleos ociosos de processadores, experimentos foram realizados com dois tipos de consultas, uma de auto-junção (C1) e outra para identificação de triângulos (C2), utilizando uma base de dados do Twitter, sob diversas dimensões de *worker nodes* e *data nodes* em um *cluster*. A Tabela 4.2 apresenta os resultados para os melhores casos dos experimentos realizados no Experimento I.

Tabela 4.2: Resultados para o Experimento I

Experimentos	Consulta C1		Consulta C2	
	↑	↓	↑	↓
Memória *Cache*	3,7%	-	4,23%	-
Sem Paralelismo de Dados	206,66%	-	244,37%	-
2 Máquinas	192,47%	-	69,54%	-
4 Máquinas	77,67%	-	27,85%	-
8 Máquinas	32,3%	-	3,67%	46,74%

Os resultados mostram que, para ambas as consultas C1 e C2, sempre há pelo menos um cenário *Avaliação* (com adição de *worker nodes*) com melhor desempenho que o cenário *Baseline*. Assim, conseguimos confirmar a hipótese do Experimento I. No melhor caso, ao processar a consulta C1, explorando todos os recursos de núcleos de processamento por meio de *worker nodes*, conseguimos aceleração de até 192,47%. Entretanto, nem sempre a adição gradativa de *worker nodes* apresentou melhores resultados; em quase todos os casos há uma deterioração a partir de um determinado ponto para a consulta C2. Isto mostra que a característica da consulta precisa ser levada em conta na escolha do melhor cenário de execução, e abre oportunidades para elaboração de heurísticas que possam ser usadas na geração do plano da consulta e definição do número de *worker nodes* a serem alocados para processar a consulta. Além disso, os resultados mostram que a memória cache teve pouca influência no tempo de processamento das consultas C1 e C2.

O próximo capítulo (Capítulo 5) apresenta o Experimento II. Este experimento avalia o comportamento dos cenários de *data nodes* e *worker nodes* no processamento de cargas de trabalho envolvendo várias consultas selecionadas do *benchmark* TPC-DS[16].

[16] http://www.tpc.org/tpcds/

CAPÍTULO 5 – EXPERIMENTO II

5.1 INTRODUÇÃO

Sistemas de gerência de Big Data recebem cargas de trabalhos com características diversas. Uma compreensão profunda das demandas imposta pela carga de trabalho é fundamental para a concepção e implementação do sistema e subsequente otimização (ZHANG et al., 2004). Dessa forma, nesse capítulo investigamos o impacto do uso de *data nodes* e *worker nodes* no processamento de cargas de trabalho.

Vários benchmarks foram desenvolvidos com a finalidade de testar o desempenho de sistemas de gerência de Big Data. Dentre estes sistemas, podemos citar YCSB (COOPER et al., 2010), BigBench (GHAZAL et al., 2013), HiBench (HUANG et al., 2010), BigDataBench (WANG, Lei et al., 2014), TPC-H[17] e TPC-DS[18]. Alguns são destinados a sistemas NoSQL (YCSB e BigDataBench), enquanto que outros são destinados a aplicações baseadas em Hadoop (TPC-DS, TPC-H, BigBench e HiBench).

Para avaliar o impacto do uso de *worker* e *data nodes* no processamento de cargas de trabalho utilizando a plataforma MyriaX, optamos por utilizar o benchmark TPC-DS. Esta escolha foi motivada por alguns fatores: trata-se de uma evolução do TPC-H para Big Data com maior complexidade e utiliza apenas dados estruturados, diferentemente dos benchmarks HiBench e BigBench.

Definida a escolha do benchmark, analisamos o conjunto de consultas do TPC-DS (99 consultas SQL no total), bem como seus fatores de escala para geração de dados. Sobre as consultas, buscamos identificar aquelas que melhor se encaixam dentro das limitações de operadores SQL que nos apresenta o MyriaX (plataforma base da execução dos experimentos). Sendo assim, selecionamos um conjunto de 11 consultas para formar a carga de trabalho a ser submetida à abordagem do uso de *worker nodes* e *data nodes* independentes em uma mesma máquina.

As máquinas do *cluster* contêm discos de dados com capacidade de 160GB e memória de 16GB. Tendo em vista estas limitações, optamos por utilizar diferentes fatores de escala para geração de dados do TPC-DS, sendo 20GB para cenários com 2 máquinas, 20GB e 40GB para cenários com 4 máquinas e 40GB e 60GB para cenários com 8 máquinas.

[17] www.tpc.org/tpch/
[18] www.tpc.org/tpcds/

O experimento descrito neste capítulo constitui-se da execução do conjunto de consultas selecionadas do TPC-DS (carga de trabalho) em vários cenários de quantidades de *worker nodes* e *data notes*, no MyriaX, com até 8 máquinas do *cluster*. Utilizamos a ferramenta de gerência de banco de dados SQL Server Management Studio[19], versão desenvolvedor, para gerar planos otimizados de execução das consultas em relação aos fatores de escala selecionados do TPC-DS. A partir destes planos, geramos manualmente os planos em formato JSON compatíveis com o MyriaX. Um algoritmo é utilizado para otimizar esta execução, capturar os tempos de 5 rodadas para cada cenário, eliminar os menores e maiores tempos e calcular a média dos tempos restantes para cada consulta e para a carga de trabalho. No restante do texto, nos referimos a esse experimento como Experimento II.

Neste capítulo, avaliamos os resultados para cada consulta individual (agrupadas por desempenho com aceleração e desaceleração), em forma de comparativo entre consultas com características semelhantes e para cargas de trabalho de cada quantidade de máquinas.

No restante do capítulo é apresentada a Visão Geral do TPC-DS (Seção 5.2), o Esquema do TPC-DS (Seção 5.3), os detalhes das Consultas (Seção 5.4), o Planejamento e Execução (Seção 5.5), os Resultados e Avaliações (Seção 5.6) e, por fim, as Considerações Finais (Seção 5.7).

5.2 VISÃO GERAL DO TPC-DS

O TPC-DS é um benchmark completo para avaliar sistemas de gerenciamento de banco de dados, agregando o TPC-H e TPC-R para sistemas de suporte à decisão (POESS et al., 2002). O TPC-DS pode ser aplicado para qualquer indústria que transforma dados externos e operacionais em inteligência de negócios (NAMBIAR; POESS, 2006). Ele modela tarefas de suporte à decisão de um típico fornecedor de produtos de varejo. O esquema, uma agregação de múltiplos esquemas estrela, contém informações de negócio tais como detalhes de cliente, vendas e dados de produtos para canais de vendas: loja, catálogo e internet. Sempre que possível, dados do mundo real são usados para popular cada tabela de forma a preservar distorções (*data skew)* de dados reais, tais como vendas sazonais e nomes frequentes (NAMBIAR; POESS, 2006).

Este benchmark abstrai uma diversidade de operações encontradas em uma aplicação de análise de informação, enquanto mantém características essenciais de desempenho (POESS et al., 2002). Uma vez que é necessário executar consultas e manutenção de dados para

[19] https://docs.microsoft.com/pt-br/sql/ssms/sql-server-management-studio-ssms

gerenciar completamente qualquer ambiente de análise de negócios, o TPC-DS contém 99 consultas distintas em SQL e 12 operações de manutenção de dados. A descrição do esquema, bem como das tabelas, é apresentada na próxima seção.

5.3 ESQUEMA DO TPC-DS

As principais características do TPC-DS incluem: múltiplos esquemas floco de neve (LEVENE; LOIZOU, 2003) com dimensões compartilhadas; 24 tabelas com uma média de 18 colunas cada; 99 consultas SQL distintas; distribuição desbalanceada de dados (*data skew*); conteúdo de banco de dados mais representativo; escala sublinear de tabelas dimensão; consultas de extração, interativas, relatórios e *ad-hoc* (NAMBIAR; POESS, 2006).

A maioria dos sistemas modernos de suporte à decisão segue a abordagem de esquema estrela para seus modelos de dados. Um esquema estrela inclui uma grande tabela fato e várias pequenas tabelas dimensões. A tabela fato armazena dados de transações realizadas frequentemente, tais como mudanças de vendas, devoluções e inventário. Cada tabela dimensão armazena adição ou mudanças de dados menos frequentes, fornecendo informações adicionais para tuplas da tabela fato, tais como o cliente que efetuou a compra. Em um esquema *snowflake,* tabelas dimensões podem ter relações com outras tabelas dimensões, além de sua relação com a tabela fato.

Como mencionado anteriormente, o TPC-DS modela funções de suporte à decisão de fornecimento de produtos de varejo. O esquema contém informações de negócios vitais tais como dados de clientes, vendas e produtos. A companhia fictícia vende bens por meio de três canais: loja (*store*), catálogo (*catalog*) e Internet (*web*). A Figura 4.1 apresenta as relações entre as sete tabelas fato, omitindo as tabelas dimensão.

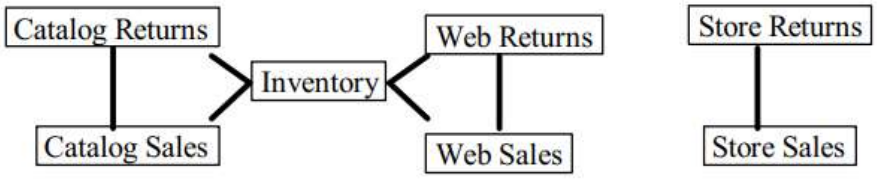

Figura 5.1: Relação entre tabelas fato do TPC-DS (POESS et al., 2002)

Cada canal de distribuição consiste de um par de tabelas fatos, modelando transações de vendas e devoluções. O número de tabelas dimensões conectadas em cada canal de distribuição (par de vendas e devoluções) é diferente. O canal *web* tem 11 tabelas dimensão enquanto que o canal *store* tem 8. Contudo, várias tabelas dimensões são compartilhadas entre os canais de distribuição com um limite de até 15 relações. Além disso, existe a tabela fato de

inventário (*inventory*) que se relaciona apenas com os canais *web* e *catalog* para rastreamento do inventário destes canais (POESS et al., 2002, p.).

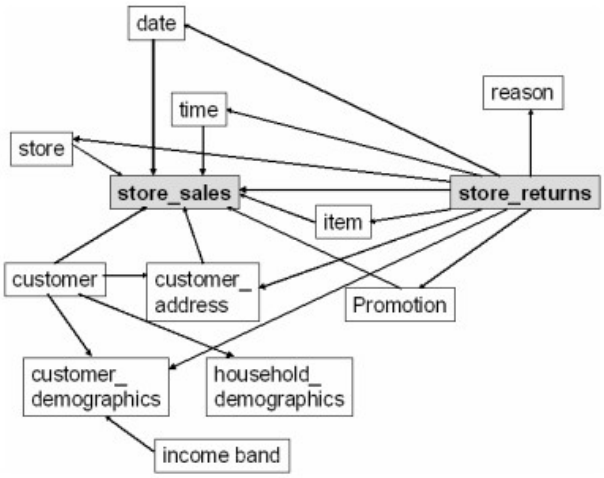

Figura 5.2: Esquema *snowflake* do canal *store* do TPC-DS (NAMBIAR; POESS, 2006)

O canal *store* consiste de duas tabelas fato, modelando vendas (*store_sales*) e devoluções (*store_returns*), como apresenta a Figura 5.2. Cada tabela fato é organizada em esquema *snowflake* com dimensões tradicionais, tais como *date_dim* (dados de data), *time_dim* (dados de tempo) e *store* (dados de loja) etc. O cliente é normalizado através das tabelas dimensão *customer, customer_address, house_demographics* e *customer_demographics* que, adicionalmente, é normalizada em *income_band*. Esta mesma constelação existe para a tabela *store_returns*. Muitas tabelas dimensão são compartilhadas entre as duas tabelas fato, enquanto que *reason* é adicionada apenas à constelação da tabela fato *store_return*. A Tabela 5.1 apresenta algumas estatísticas sobre as tabelas do TPC-DS.

Tabela 5.1: Estatísticas do esquema do TPC-DS (NAMBIAR; POESS, 2006)

Número de tabelas fato	7	
Número de tabelas dimensão	17	
Número de colunas	**Min**	3
	Max	34
	Med	18
Número de relações	104	

Quanto ao conjunto de dados, o TPC-DS utiliza uma abordagem híbrida de escala de dados e domínios. Enquanto geradores de dados sintéticos têm ótimas vantagens, este benchmark segue uma abordagem híbrida que mescla dados do mundo real e dados sintéticos baseado em domínios de dados (NAMBIAR; POESS, 2006). Ele escala em termos de fatores discretos (GB): 100, 200, 1.000, 3.000, 10.000, 30.000 e 100.000. A Tabela 5.2 apresenta cardinalidades de algumas tabelas fato e dimensão referente a algumas destas escalas.

Tabela 5.2: Cardinalidades de tabelas ($K=10^3$, $M=10^6$, $G=10^9$) (NAMBIAR; POESS, 2006)

Tabelas	Número de linhas						
	20GB	40GB	60GB	100GB	1TB	10TB	100TB
store_sales	57M	115M	172M	288M	2,9G	30G	297G
store_returns	5M	11M	17M	14M	147M	1,5G	15G
store	44	112	178	200	500	750	1,500
customer	266K	600K	933K	2M	8M	20M	100M
item	28K	52K	74K	200K	300K	400K	500K

Com base na Tabela 5.2, utilizamos os fatores de escala de 20GB, 40GB e 60GB para gerar as bases de dados utilizadas no experimento apresentado neste capítulo. Uma visão geral das consultas do TPC-DS é apresentada na seção seguinte.

5.4 CONSULTAS

O TPC-DS possui um conjunto amplo com 99 consultas distribuídas em quatro classes: relatórios (41), *ad-hoc* (59), interativa (4) e mineração de dados (23). A Tabela 5.3 apresenta estatísticas referentes à cobertura do esquema e a Tabela 5.4 apresenta estatísticas referentes às tabelas do TPC-DS.

Tabela 5.3: Estatísticas de consultas por cobertura do esquema (POESS et al., 2007)

Cobertura do Esquema	Número que consultas
Uma tabela fato	54
Várias tabelas fato	39
Apenas tabelas dimensão	6
Apenas canal de venda *store*	37
Apenas canal de venda *catalog*	12
Apenas canal de venda *web*	12

Tabela 5.4: Estatísticas de consultas em relação às tabelas (POESS et al., 2007)

Tabela	Número que consultas
date_dim	91
store	68
store_sales	64
item	58
customer	57
catalog_sales	38
web_sales	36
customer_address	34
customer_demographics	20
household_demographics	17
store_returns	15
catalog_returns	12
Promotion	10
warehouse	10
web_returns	10
call_center	6
income_band	6
time_dim	5
web_site	5

Tabela 5.5: Estatísticas de consultas por recurso SQL (POESS et al., 2007)

Recurso SQL	Número que consultas
Subexpressão comum	31
Subconsulta correlacionada	15
Subconsultas não correlacionadas	76
Group By	78
Order By	64
Rollup	9
Partition	11
Exists	5
União	17
Interseção	2
Mínimo	1
Caso	24
Contém	5

As consultas do TPC-DS contêm uma grande diversidade de recursos e funcionalidades, como apresenta a Tabela 5.5. Dentre o conjunto de 99 consultas geradas de forma automática em formato SQL através da ferramenta de geração de consultas deste benchmark, selecionamos 11 considerando a complexidade das consultas e a limitação dos operadores reconhecidos pelo mecanismo de execução de consulta MyriaX. Neste contexto, foram selecionadas consultas com alta complexidade, que abrangem tabelas fato e dimensões, que utilizam operadores relacionais reconhecidos pelo MyriaX. Os motivos de descarte das demais consultas são apresentados no ANEXO A.

Após a escolha do conjunto de 11 consultas, tornou-se necessário gerar os planos de execução paralela para estas consultas em formato JSON (formato utilizado pelo MyriaX). Nesta etapa, algumas modificações no código da consulta foram necessárias para adequar ao plano de execução em formato JSON. Tais modificações, no entanto, não alteraram a estrutura principal de processamento das consultas. As consultas, bem como as modificações realizadas, são descritas a seguir.

CONSULTA 7: esta consulta é aplicada sobre o canal *Store* e utiliza 1 tabela fato (*store_sales*) e 4 dimensões (*customer_demographics, date_dim, store e item*). O código desta consulta é apresentado na Figura 5.3.

Para esta consulta, foram necessárias modificações na linha 3, 13 e 15. A linha 3 foi modificada para “s_state” pois o MyriaX não reconhece o operador “*grouping*”. A linha 17 foi modificada para “**and** s_state = ‘TN’” pois tratava-se de uma comparação com uma lista de valores iguais. Por fim, a linha 19 foi modificada para “i_item_id, s_state” pois o MyriaX não reconhece o operador “*rollup*”.

```
1   select
2       i_item_id,
3       s_state, grouping(s_state) g_state,
4       avg(ss_quantity) agg1, avg(ss_list_price) agg2,
5       avg(ss_coupon_amt) agg3, avg(ss_sales_price) agg4
6   from
7       store_sales, customer_demographics, date_dim, store, item
8   where
9       ss_sold_date_sk = d_date_sk and ss_item_sk = i_item_sk
10      and ss_store_sk = s_store_sk and ss_cdemo_sk = cd_demo_sk
11      and cd_gender = 'F' and cd_marital_status = 'W'
12      and cd_education_status = 'Primary' and d_year = 1998
13      and s_state in ('TN','TN', 'TN', 'TN', 'TN', 'TN')
14  group by
15      rollup (i_item_id, s_state)
16  order by
17      i_item_id, s_state
```

Figura 5.3: Código SQL da consulta 7 do TPC-DS

CONSULTA 15: esta consulta é aplicada sobre o canal *Catalog* e utiliza 1 tabela fato (*catalog_sales*) e 5 dimensões (*customer_demographics, customer, customer_address, date_dim e item*). O código desta consulta é apresentado na Figura 5.4.

```
1   select
2       i_item_id, ca_country, ca_state, ca_county,
3       avg(cast(cs_quantity as decimal(12,2))) agg1,
4       avg(cast(cs_list_price as decimal(12,2))) agg2,
5       avg(cast(cs_coupon_amt as decimal(12,2))) agg3,
6       avg(cast(cs_sales_price as decimal(12,2))) agg4,
7       avg(cast(cs_net_profit as decimal(12,2))) agg5,
8       avg(cast(c_birth_year as decimal(12,2))) agg6,
9       avg(cast(cd1.cd_dep_count as decimal(12,2))) agg7
10  from
11      catalog_sales, customer_demographics cd1, customer_demographics cd2, customer,
12      customer_address, date_dim, item
13  where
14      cs_sold_date_sk = d_date_sk and cs_item_sk = i_item_sk
15      and cs_bill_cdemo_sk = cd1.cd_demo_skand cs_bill_customer_sk = c_customer_sk
16      and cd1.cd_gender = 'F' and cd1.cd_education_status = 'Advanced Degree'
17      and c_current_cdemo_sk = cd2.cd_demo_sk and c_current_addr_sk = ca_address_sk
18      and c_birth_month in (2,10,5,7,8,3) and d_year = 2001
19      and ca_state in ('GA','TN','WI','TX','NY','MS','AR')
20  group by
21      rollup (i_item_id, ca_country, ca_state, ca_county)
22  order by
23      ca_country, ca_state, ca_county, i_item_id
```

Figura 5.4: Código SQL da consulta 15 do TPC-DS

Para esta consulta, foi necessário remover o operador ***cast*** das linhas 3-9, pois o MyriaX não reconhece este operador, e modificar a linha 21. A linha 26 foi modificada para "i_item_id, ca_country, ca_state, ca_county" pois o MyriaX não reconhece o operador "*rollup*".

CONSULTA 17: esta consulta é aplicada sobre o canal *Catalog* e utiliza 1 tabela fato (*catalog_returns*) e 6 dimensões (*call_center, date_dim, customer, customer_address,*

customer_demographics e household_demographics). O código desta consulta é apresentado na Figura 5.5.

```
1   select
2       cc_call_center_id Call_Center, cc_name Call_Center_Name, cc_manager Manager,
3       sum(cr_net_loss) Returns_Loss
4   from
5       call_center, catalog_returns, date_dim, customer, customer_address, customer_demographics,
6       household_demographics
7   where
8       cr_call_center_sk = cc_call_center_sk and cr_returned_date_sk = d_date_sk
9       and cr_returning_customer_sk = c_customer_sk and cd_demo_sk = c_current_cdemo_sk
10      and hd_demo_sk = c_current_hdemo_sk and ca_address_sk = c_current_addr_sk
11      and d_year = 2001 and d_moy = 12
12      and ((cd_marital_status = 'M' and cd_education_status = 'Unknown') or (cd_marital_status =
13      'W' and cd_education_status = 'Advanced Degree'))
14      and hd_buy_potential like '501-1000%'
15      and ca_gmt_offset = -6
16  group by
17      cc_call_center_id, cc_name, cc_manager, cd_marital_status, cd_education_status
18  order by
19      sum(cr_net_loss) desc
```

Figura 5.5: Código SQL da consulta 17 do TPC-DS

Para esta consulta, foi necessário realizar modificação na linha 14. A linha 14 foi modificada para "**and SUBSTRING**(hd_buy_potential,1,8) = '501-1000'" pois o MyriaX não reconhece o operador "*like*".

CONSULTA 19: esta consulta é aplicada sobre o canal *Catalog* e utiliza 1 tabela fato (*catalog_returns*) e 6 dimensões (*call_center, date_dim, customer, customer_address, customer_demographics e household_demographics*). O código desta consulta é apresentado na Figura 5.6. Para esta consulta, modificações não foram necessárias.

```
1   select
2       count(*)
3   from
4       store_sales, household_demographics, time_dim, store
5   where
6       ss_sold_time_sk = time_dim.t_time_sk and ss_hdemo_sk = household_demographics.hd_demo_sk
7       and ss_store_sk = s_store_sk and time_dim.t_hour = 20 and time_dim.t_minute >= 30
8       and household_demographics.hd_dep_count = 8 and store.s_store_name = 'ese'
9   order by
10      count(*)
```

Figura 5.6: Código SQL da consulta 19 do TPC-DS

CONSULTA 25: esta consulta é aplicada sobre os canais *Store* e *Catalog* e utiliza 3 tabelas fato (*store_sales, store_returns* e *catalog_sales*) e 3 dimensões (*date_dim, store* e *item*). O código desta consulta é apresentado na Figura 5.7.

```
1   select
2       i_item_id, i_item_desc, s_store_id, s_store_name, avg(ss_quantity) as store_sales_quantity,
3       avg(sr_return_quantity) as store_returns_quantity, avg(cs_quantity) as catalog_sales_quantity
4   from
5       store_sales, store_returns, catalog_sales, date_dim d1, date_dim d2, date_dim d3, store, item
6   where
7       d1.d_moy = 4 and d1.d_year = 2000 and d1.d_date_sk = ss_sold_date_sk
8       and i_item_sk = ss_item_sk and s_store_sk = ss_store_sk
9       and ss_customer_sk = sr_customer_sk and ss_item_sk = sr_item_sk
10      and ss_ticket_number = sr_ticket_number and sr_returned_date_sk = d2.d_date_sk
11      and d2.d_moy between 4 and 4 + 3
12      and d2.d_year = 2000 and sr_customer_sk = cs_bill_customer_sk and sr_item_sk = cs_item_sk
13      and cs_sold_date_sk = d3.d_date_sk
14      and d3.d_year in (2000, 2000+1, 2000+2)
15  group by
16      i_item_id, i_item_desc, s_store_id, s_store_name
17  order by
18      i_item_id, i_item_desc, s_store_id, s_store_name
```

Figura 5.7: Código SQL da consulta 25 do TPC-DS

Para esta consulta, foi necessário realizar modificações nas linhas 11 e 14 para evitar operações de soma desnecessárias. A linha 11 foi modificada para "**and** d2.d_moy **between** 4 **and** 7". Da mesma forma, a linha 14 foi modificada para "**and** d3.d_year **in** (2000, 2001, 2002)".

CONSULTA 26: esta consulta é aplicada sobre o canal *Store* e utiliza 1 tabela fato (*store_sales*) e 4 dimensões (*store, customer_demographics, customer_address e date_dim*). O código desta consulta é apresentado na Figura 5.8. Para esta consulta, modificações não foram necessárias.

```
1   select
2       sum(ss_quantity)
3   from
4       store_sales, store, customer_demographics, customer_address, date_dim
5   where
6       s_store_sk = ss_store_sk and ss_sold_date_sk = d_date_sk and d_year = 1999
7       and ((cd_demo_sk = ss_cdemo_sk and cd_marital_status = 'M'
8           and cd_education_status = '4 yr Degree' and ss_sales_price between 100.00
9           and 150.00)
10          or (cd_demo_sk = ss_cdemo_sk and cd_marital_status = 'S'
11              and cd_education_status = 'College' and ss_sales_price between 50.00 and
12              100.00)
13          or (cd_demo_sk = ss_cdemo_sk and cd_marital_status = 'W'
14              and cd_education_status = '2 yr Degree' and ss_sales_price between 150.00
15              and 200.00))
16      and ((ss_addr_sk = ca_address_sk and ca_country = 'United States'
17          and ca_state in ('TN', 'KS', 'MO') and ss_net_profit between 0 and 2000)
18          or (ss_addr_sk = ca_address_sk and ca_country = 'United States'
19              and ca_state in ('VA', 'MI', 'IL') and ss_net_profit between 150 and 3000)
20          or (ss_addr_sk = ca_address_sk and ca_country = 'United States'
21              and ca_state in ('KY', 'OH', 'TX') and ss_net_profit between 50 and 25000))
```

Figura 5.8: Código SQL da consulta 26 do TPC-DS

CONSULTA 27: esta consulta é aplicada sobre os canais *Catalog* e utiliza 1 tabela fato (*catalog_sales*) e 3 dimensões (*customer, customer_address* e *date_dim*). O código desta consulta é apresentado na Figura 5.9.

```
1   select
2       ca_zip, sum(cs_sales_price)
3   from
4       catalog_sales, customer, customer_address, date_dim
5   where
6       cs_bill_customer_sk = c_customer_sk
7       and c_current_addr_sk = ca_address_sk
8       and (
9               SUBSTRING(ca_zip,1,5) in ('85669', '86197','88274','83405','86475','85392', '85460', '80348', '81792')
10              or ca_state in ('CA','WA','GA') or cs_sales_price > 500)
11      and cs_sold_date_sk = d_date_sk and d_qoy = 2  and d_year = 2000
12  group by
13      ca_zip
14  order by
15      ca_zip
```

Figura 5.9: Código SQL da consulta 27 do TPC-DS

Para esta consulta, foi realizada modificação na linha 9. A linha 9 foi modificada para "ca_zip **in** ('85669', '86197', '88274', '83405', '86475', '85392', '85460', '80348', '81792')" pois os valores do atributo "*ca_zip*" têm 5 dígitos, o que torna desnecessário o uso da operação "*substring*".

CONSULTA 29: esta consulta é aplicada sobre o canal *Store* e utiliza 1 tabela fato (*store_sales*) e 4 dimensões (*customer_demographics, date_dim, item* e *promotion*). O código desta consulta é apresentado na Figura 5.10. Para esta consulta, modificações não foram necessárias.

```
1   select
2       i_item_id, avg(ss_quantity) agg1, avg(ss_list_price) agg2, avg(ss_coupon_amt)
3       agg3, avg(ss_sales_price) agg4
4   from
5       store_sales, customer_demographics, date_dim, item, promotion
6   where
7       ss_sold_date_sk = d_date_sk and ss_item_sk = i_item_sk
8       and ss_cdemo_sk = cd_demo_sk and ss_promo_sk = p_promo_sk
9       and cd_gender = 'F' and cd_marital_status = 'W'
10      and cd_education_status = 'Advanced Degree'
11      and (p_channel_email = 'N' or p_channel_event = 'N') and d_year = 2000
12  group by
13      i_item_id
14  order by
15      i_item_id
```

Figura 5.10: Código SQL da consulta 29 do TPC-DS

CONSULTA 48: esta consulta é aplicada sobre o canal *Catalog* e utiliza 1 tabela fato (*catalog_sales*) e 4 dimensões (*customer_demographics, date_dim, item* e *promotion*). O

código desta consulta é apresentado na Figura 5.11. Para esta consulta, modificações não foram necessárias.

```
1   select
2       i_item_id, avg(cs_quantity) agg1, avg(cs_list_price) agg2, avg(cs_coupon_amt) agg3,
3       avg(cs_sales_price) agg4
4   from
5       catalog_sales, customer_demographics, date_dim, item, promotion
6   where
7       cs_sold_date_sk = d_date_sk and cs_item_sk = i_item_sk
8       and cs_bill_cdemo_sk = cd_demo_sk and cs_promo_sk = p_promo_sk
9       and cd_gender = 'F' and cd_marital_status = 'D' and cd_education_status = 'Advanced Degree'
10      and (p_channel_email = 'N' or p_channel_event = 'N') and d_year = 1998
11  group by
12      i_item_id
13  order by
14      i_item_id
```

Figura 5.11: Código SQL da consulta 48 do TPC-DS

CONSULTA 91: esta consulta é aplicada sobre o canal *Store* e utiliza 1 tabela fato (*store_sales*) e 5 dimensões (*date_dim, item, customer, customer_address* e *store*). O código desta consulta é apresentado na Figura 5.12.

```
1   select
2       i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact, sum(ss_ext_sales_price) ext_price
3   from
4       store_sales, date_dim, item, customer, customer_address, store
5   where
6       d_date_sk = ss_sold_date_sk and ss_item_sk = i_item_sk
7       and i_manager_id=25 and d_moy=11 and d_year=2001
8       and ss_customer_sk = c_customer_sk and c_current_addr_sk = ca_address_sk
9       and SUBSTRING (ca_zip,1,5) <> SUBSTRING (s_zip,1,5)
10      and ss_store_sk = s_store_sk
11  group by
12      i_brand, i_brand_id, i_manufact_id, i_manufact
13  order by
14      ext_price desc, i_brand, i_brand_id, i_manufact_id, i_manufact
```

Figura 5.12: Código SQL da consulta 91 do TPC-DS

Para esta consulta, foi realizada modificação na linha 9. A linha 9 foi modificada para "**and** ca_zip <> s_zip" pois, da mesma forma que na Consulta 27, a operação de "*substring*" não é necessária tendo em vista que os valores destes atributos têm 5 dígitos.

CONSULTA 96: esta consulta é aplicada sobre os canais *Store* e *Catalog* e utiliza 3 tabelas fato (*store_sales, store_returns* e *catalog_sales*) e 3 dimensões (*date_dim, store* e *item*). O código desta consulta é apresentado na Figura 5.13. Para esta consulta, modificações não foram necessárias.

```
1   select
2       i_item_id, i_item_desc, s_store_id, s_store_name, sum(ss_net_profit) as store_sales_profit,
3       sum(sr_net_loss) as store_returns_loss, sum(cs_net_profit) as catalog_sales_profit
4   from
5       store_sales, store_returns, catalog_sales, date_dim d1, date_dim d2, date_dim d3, store, item
6   where
7       d1.d_moy = 4 and d1.d_year = 2000 and d1.d_date_sk = ss_sold_date_sk
8       and i_item_sk = ss_item_sk and s_store_sk = ss_store_sk
9       and ss_customer_sk = sr_customer_sk and ss_item_sk = sr_item_sk
10      and ss_ticket_number = sr_ticket_number and sr_returned_date_sk = d2.d_date_sk
11      and d2.d_moy between 4 and 10 and d2.d_year = 2000
12      and sr_customer_sk = cs_bill_customer_sk and sr_item_sk = cs_item_sk
13      and cs_sold_date_sk = d3.d_date_sk and d3.d_moy between 4 and  10 and d3.d_year = 2000
14  group by
15      i_item_id, i_item_desc, s_store_id, s_store_name
16  order by
17      i_item_id, i_item_desc, s_store_id, s_store_name
```

Figura 5.13: Código SQL da consulta 96 do TPC-DS

Deste modo, a carga de trabalho do experimento apresentado neste capítulo é composta pelo conjunto de 11 consultas apresentadas acima. A próxima seção apresenta o planejamento e execução do experimento apresentado neste capítulo.

5.5 PLANEJAMENTO E EXECUÇÃO

Esta seção apresenta os detalhes do planejamento e execução do Experimento II no mesmo *cluster* utilizado no Experimento I. Os detalhes apresentados são referentes a configuração de *data* e *worker nodes*, base de dados, planos de execução das consultas e *script* de execução do experimento.

5.5.1 CONFIGURAÇÃO DE DATA E WORKER NODES

Neste experimento também definidos a quantidade de *data nodes* como sendo igual à quantidade de discos de dados disponíveis e a quantidade de *worker nodes* tendo como limite a quantidade de núcleos de processamento disponíveis, excluindo-se os núcleos alocados para *data nodes*. Cada cenário, assim como no Experimento I, é composto por uma quantidade de máquinas (*m*), *data nodes* (*dn*) e *worker nodes* (*wn*), onde a quantidade de *worker nodes*, quando maior que zero, é dividida igualmente para cada máquina do cenário. A Tabela 5.6 apresenta a distribuição de tais cenários de 2 até 8 máquinas. A heurística para criar os cenários foi a mesma do Experimento I. Neste experimento, *data nodes*, além de realizar operações de leitura e escrita de dados, também atuam no *pipeline* de processamento dos operadores relacionais como *worker nodes*.

Tabela 5.6: Cenários para até 8 máquinas com 8 núcleos de processamento e 1 disco de dados cada

Cenário	# máquinas	# *data nodes*	# *worker nodes*	# nós de processamento
m:2_dn:2_wn:0	4	2	0	2
m:2_dn:2_wn:2	4	2	2	4
m:2_dn:2_wn:6	4	2	6	8
m:2_dn:2_wn:14	4	2	14	16
m:4_dn:4_wn:0	4	4	0	4
m:4_dn:4_wn:4	4	4	4	8
m:4_dn:4_wn:12	4	4	12	16
m:4_dn:4_wn:28	4	4	28	32
m:8_dn:8_wn:0	8	8	0	8
m:8_dn:8_wn:8	8	8	8	16
m:8_dn:8_wn:24	8	8	24	32
m:8_dn:8_wn:56	8	8	56	64

A Tabela 5.6 apresenta os cenários possíveis para implantações em até 8 máquinas, destacando em cinza os cenários padrão do MyriaX. Nesta tabela, a coluna "*# nós de processamento*", assim como no Experimento I, contabiliza a quantidade de nós atuando no *pipeline* de processamento (*data nodes* + *worker nodes*). A quantidade máxima de memória principal utilizada por cada nó é a memória total que cada máquina oferece (16GB) dividida igualmente pela quantidade de nós alocados em cada máquina. Sendo assim, quando são alocados 8 nós em uma máquina, cada nó pode utilizar no máximo 2GB de memória principal. Em todos os cenários, há o acréscimo de uma máquina para atuar como *master node* e, portanto, apenas as demais máquinas são consideradas pelos cenários. Cabe ressaltar que os dados são particionados entre os *data nodes* usando a estratégia de *round-robin* (MEHTA; DEWITT, 1997).

5.5.2 BASE DE DADOS

Durante a execução do Experimento II, realizamos o processamento da carga de trabalho usando as 11 consultas selecionadas do TPC-DS (descritas na Seção 5.4) sobre os cenários apresentados na Tabela 5.6, utilizando a base de dados com três fatores de escala, sendo um fator para cada quantidade de máquina. Esta variação foi necessária devido às limitações de espaço em disco e memória do *cluster*. A identificação de cada base de dados, bem como os fatores de escala é apresentada na Tabela 5.7. Após a geração das bases de dados, tabulamos tamanhos e quantidades de tuplas de cada tabela em relação a cada base de dados. Esta relação é apresentada na Tabela 5.8.

Tabela 5.7: Relação entre quantidade de máquinas e fatores de geração da base de dados do TPC-DS

# Máquinas	Identificação	Fator de escala
2	Base 1	20GB
4	Base 2	40GB
8	Base 3	60GB

Tabela 5.8: Relação entre tamanho e quantidade de tuplas de cada tabela para cada base de dados gerada

Tabelas	Tamanho			# Tuplas		
	Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
call_center	4KB	4KB	4KB	6	8	8
catalog_returns	417MB	842MB	1,7GB	2.882.133	5.759.645	8.640.070
catalog_sales	5,6GB	12GB	23GB	28.803.583	57.596.009	86.401.774
customer_address	14MB	32MB	67MB	133.000	300.000	466.000
customer	34MB	77MB	163MB	266.000	600.000	933.000
customer_demographics	76MB	76MB	76MB	1.920.800	1.920.800	1.920.800
date_dim	9,8MB	9,8MB	9,8MB	73.049	73.049	73.049
household_demographics	144KB	144KB	144KB	7.200	7.200	7.200
item	7,6MB	14MB	26MB	28.000	52.000	74.000
promotion	44KB	60KB	84KB	355	466	577
store	12KB	32KB	64KB	44	112	178
store_returns	640MB	1,3GB	2,6GB	5.755.095	11.514.645	17.271.432
store_sales	7,4GB	15GB	31GB	57.598.932	115.203.420	172.800.711
time_dim	4,9MB	4,9MB	4,9MB	86.400	86.400	86.400
web_sales	2,8GB	5,7GB	12GB	14.396.103	28.797.094	43.198.797

5.5.3 PLANOS DE EXECUÇÃO

Com base nas consultas apresentadas na Seção 5.4, utilizamos a ferramenta SQL Server Management Studio para gerar planos de execução otimizados para as 11 consultas selecionadas do TPC-DS. Esta ferramenta se baseia na base de dados e no código SQL da consulta para gerar um plano de execução otimizado, visando obter melhor desempenho para a consulta. Estes planos foram convertidos para o formato JSON para serem reconhecidos e processados pelo MyriaX de forma paralela durante a execução do Experimento II.

Nesta seção, são apresentadas as versões gráficas destes planos, onde os valores de cardinalidades e resultados das operações internas de cada plano são referentes à Base 3 apresentada na Seção 5.5.2: Consulta 7 (Figura 5.14), Consulta 15 (Figura 5.15), Consulta 17 (Figura 5.16), Consulta 19 (Figura 5.17), Consulta 25 (Figura 5.18), Consulta 26 (Figura 5.19), Consulta 27 (Figura 5.20), Consulta 29 (Figura 5.21), Consulta 48 (Figura 5.22), Consulta 91 (Figura 5.23) e Consulta 96 (Figura 5.24). As cardinalidades destas figuras utilizam as constantes K = 10^3, M = 10^6, G = 10^9 e T = 10^{12}.

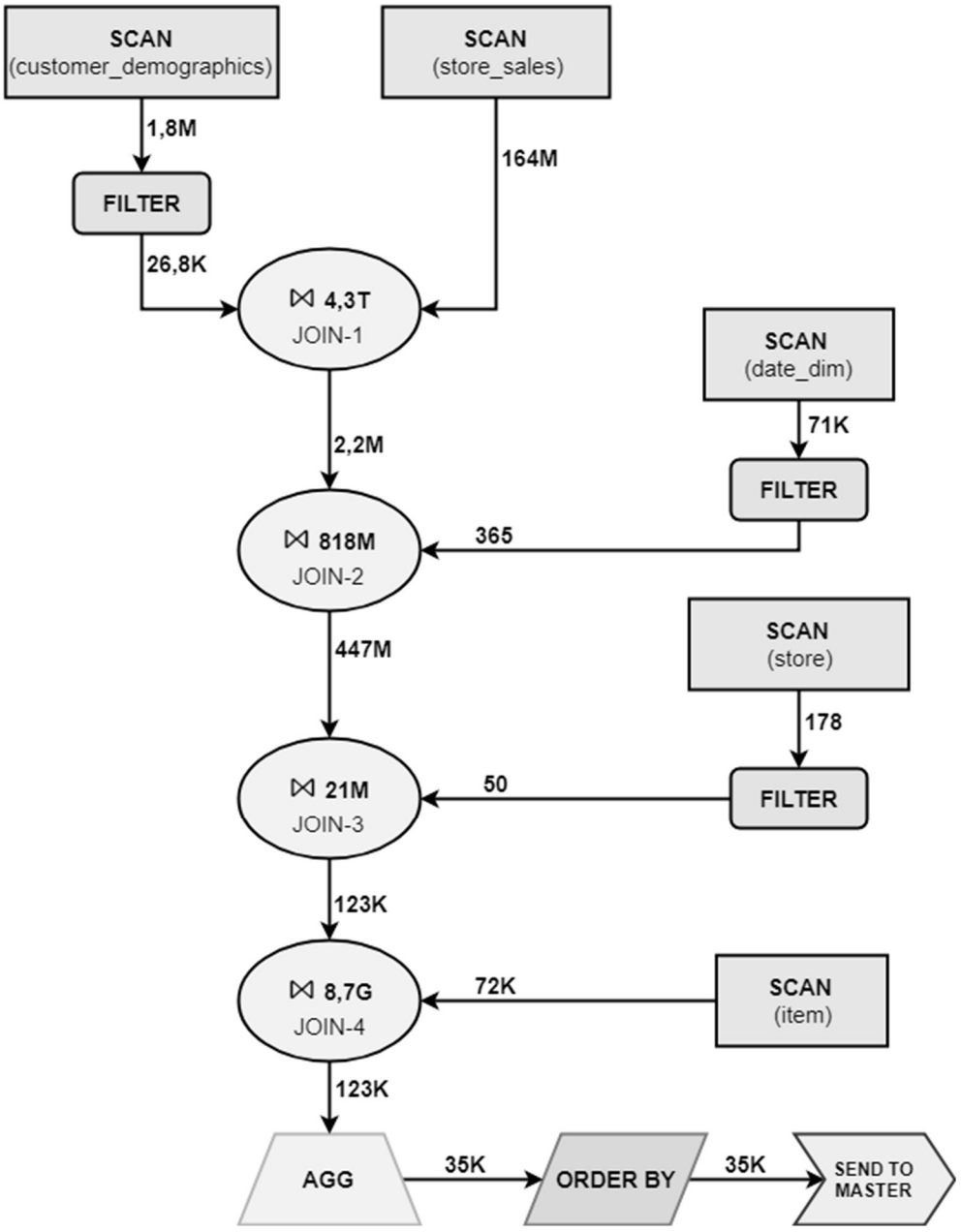

Figura 5.14: Plano de execução da Consulta 7

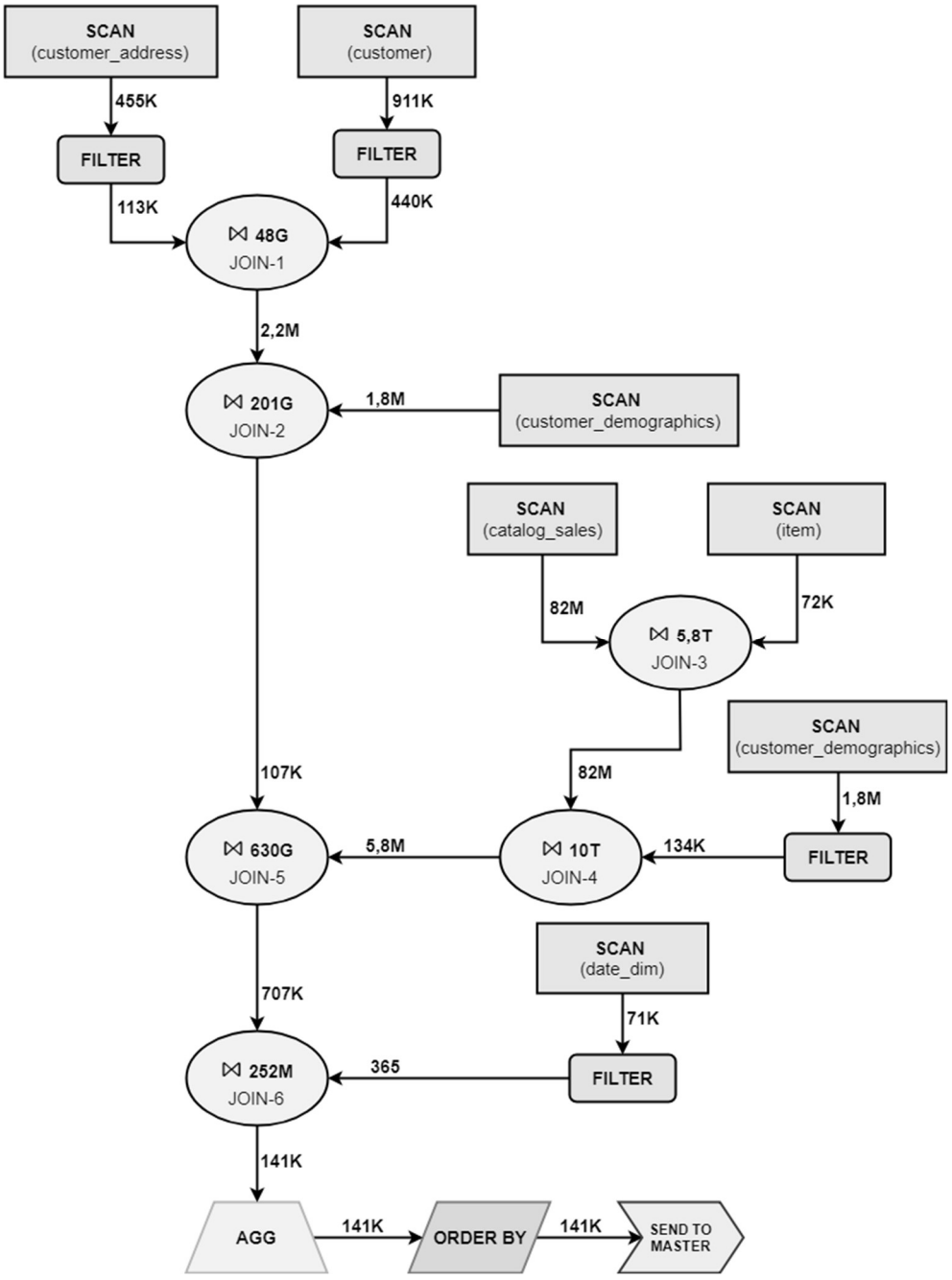

Figura 5.15: Plano de execução da Consulta 15

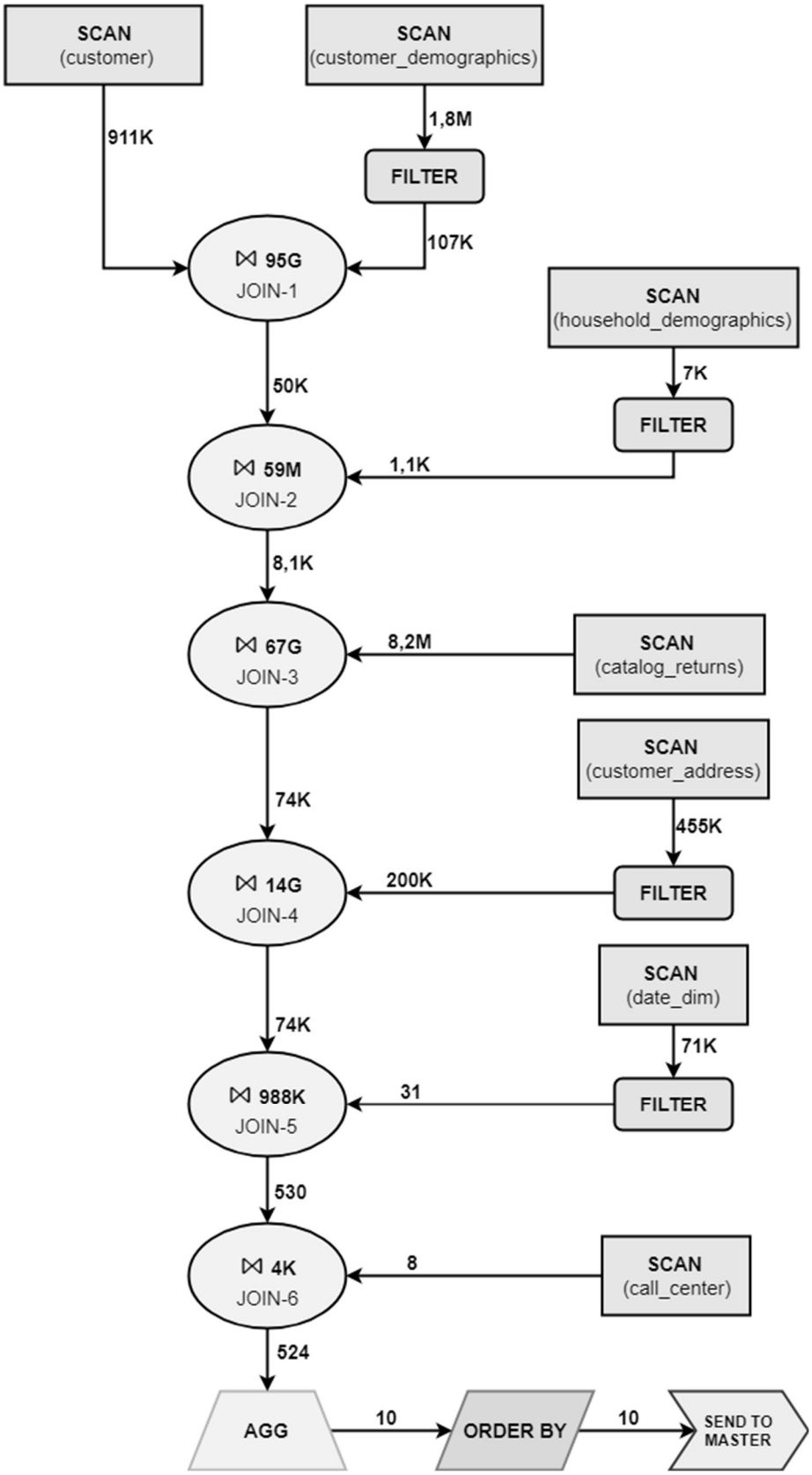

Figura 5.16: Plano de execução da Consulta 17

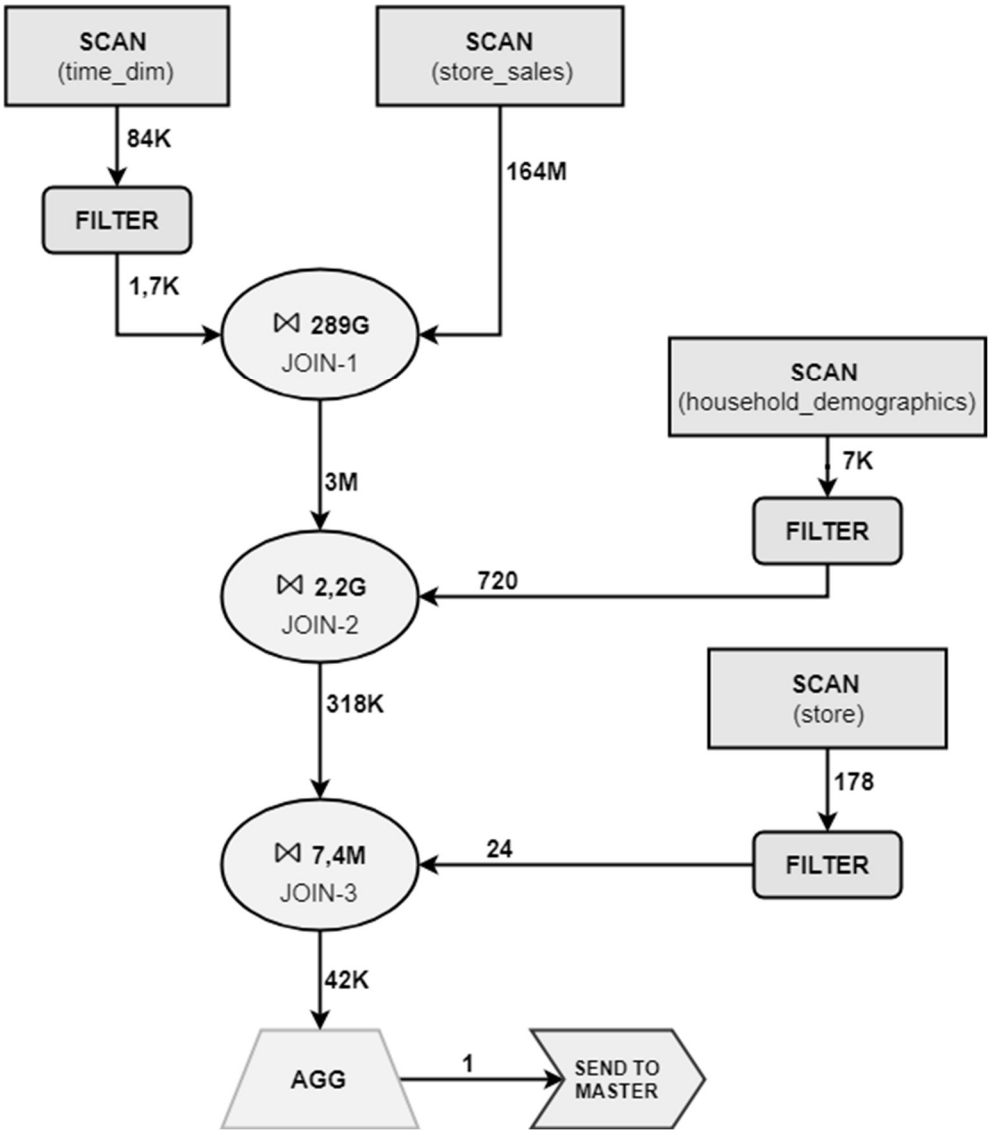

Figura 5.17: Plano de execução da Consulta 19

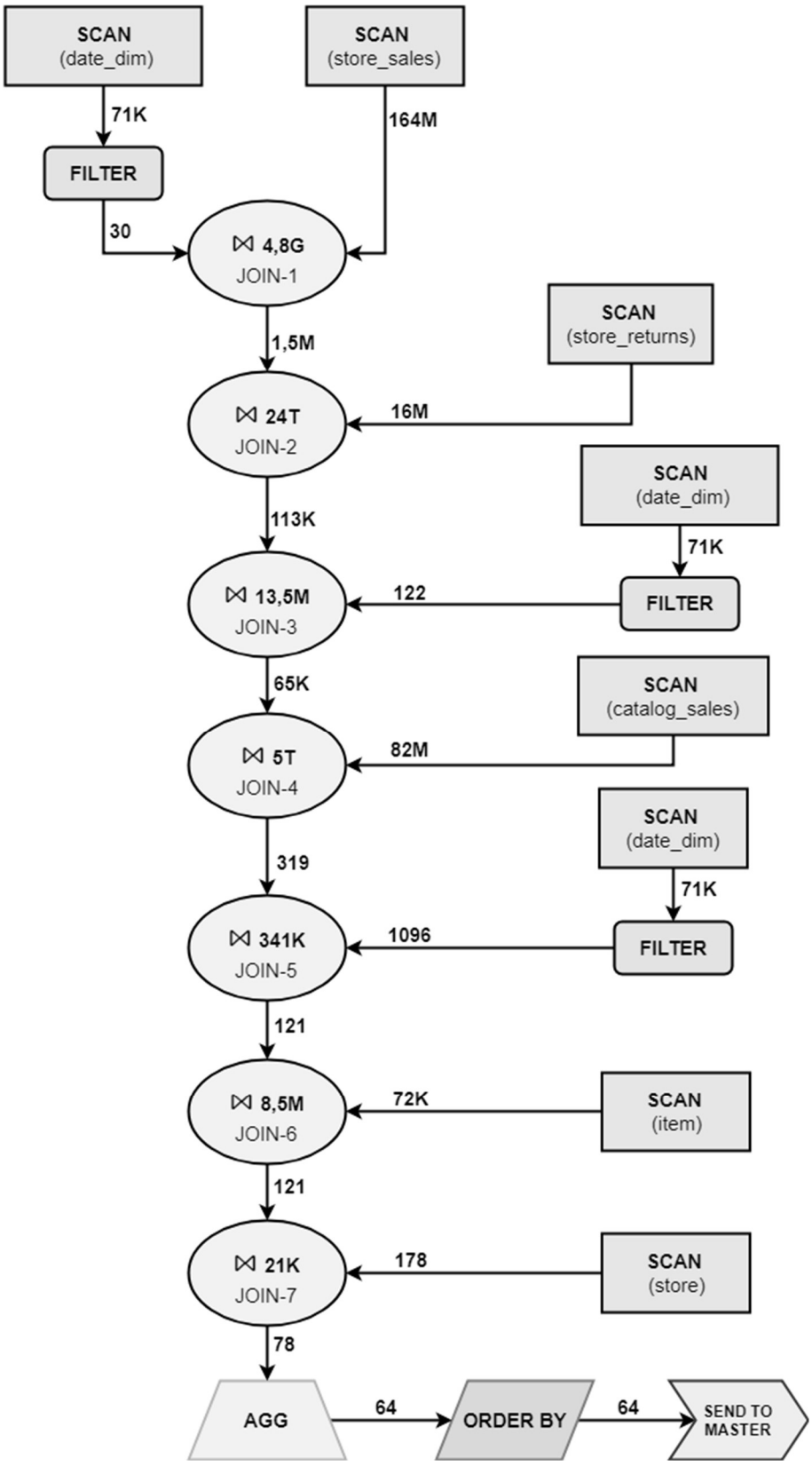

Figura 5.18: Plano de execução da Consulta 25

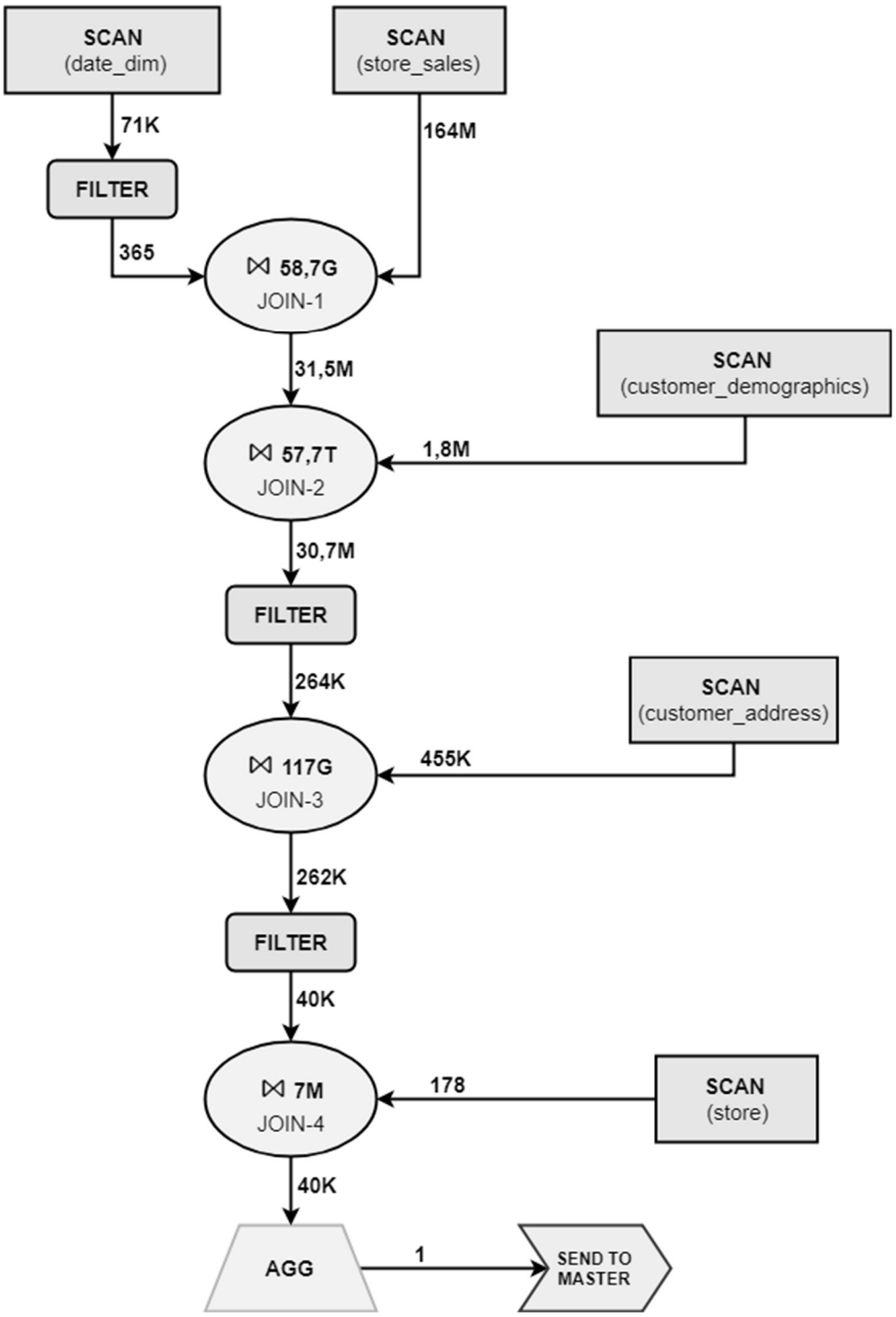

Figura 5.19: Plano de execução da Consulta 26

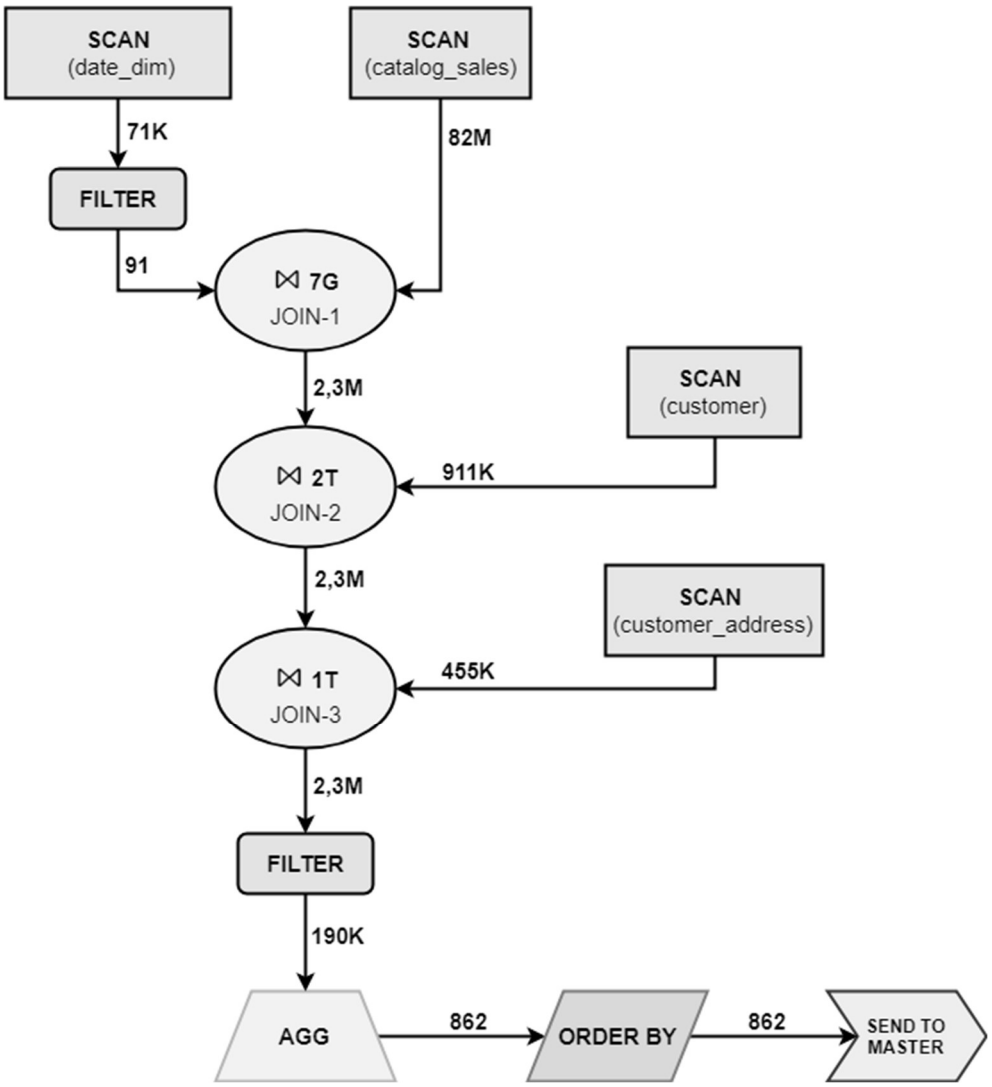

Figura 5.20: Plano de execução da Consulta 27

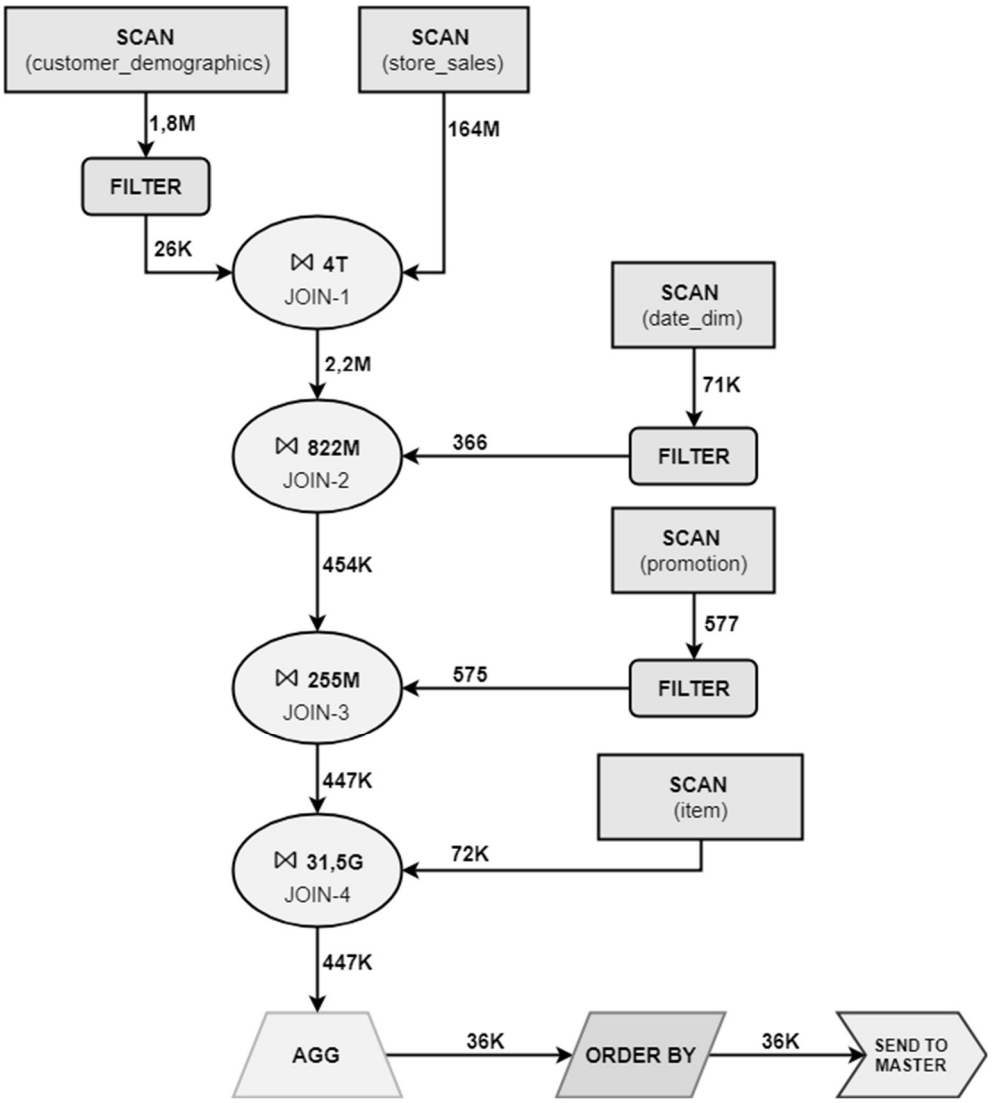

Figura 5.21: Plano de execução da Consulta 29

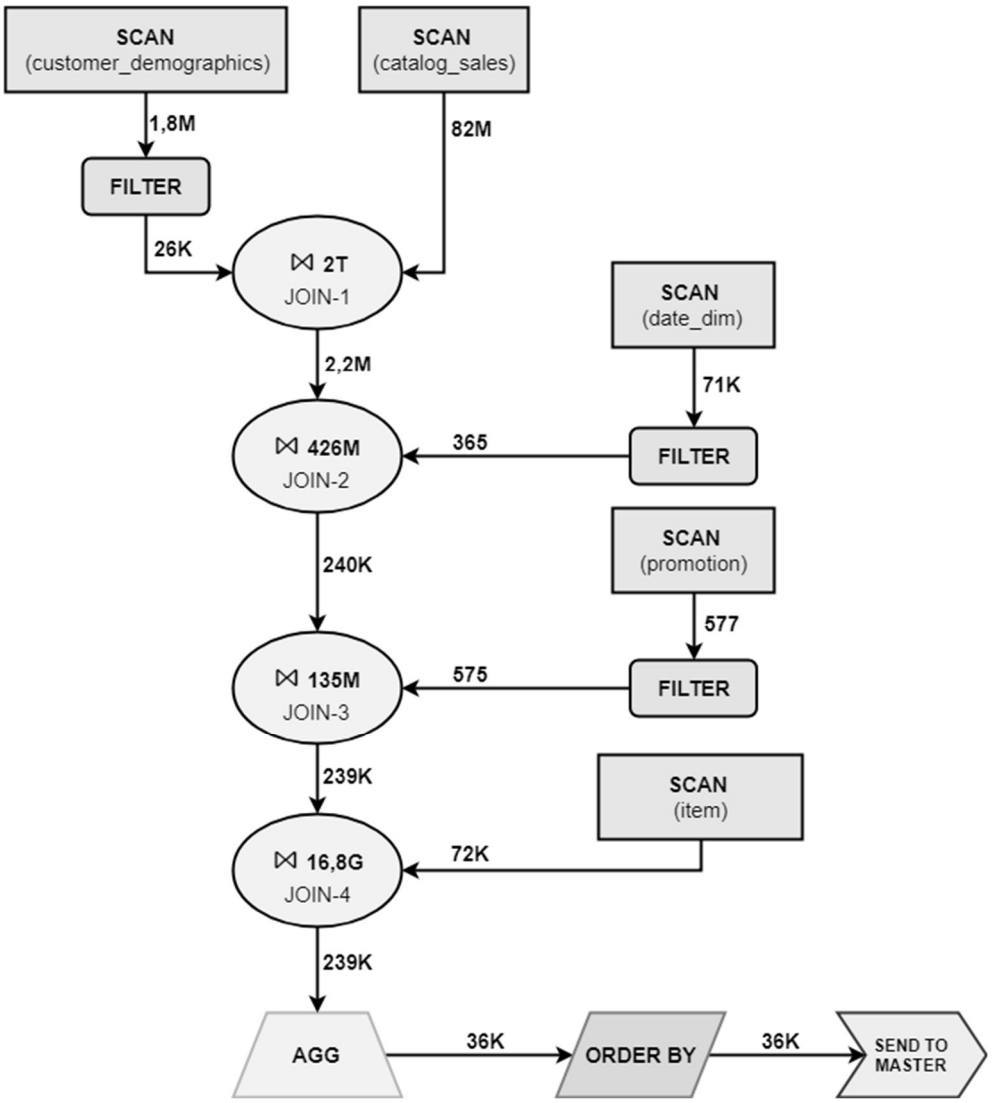

Figura 5.22: Plano de execução da Consulta 48

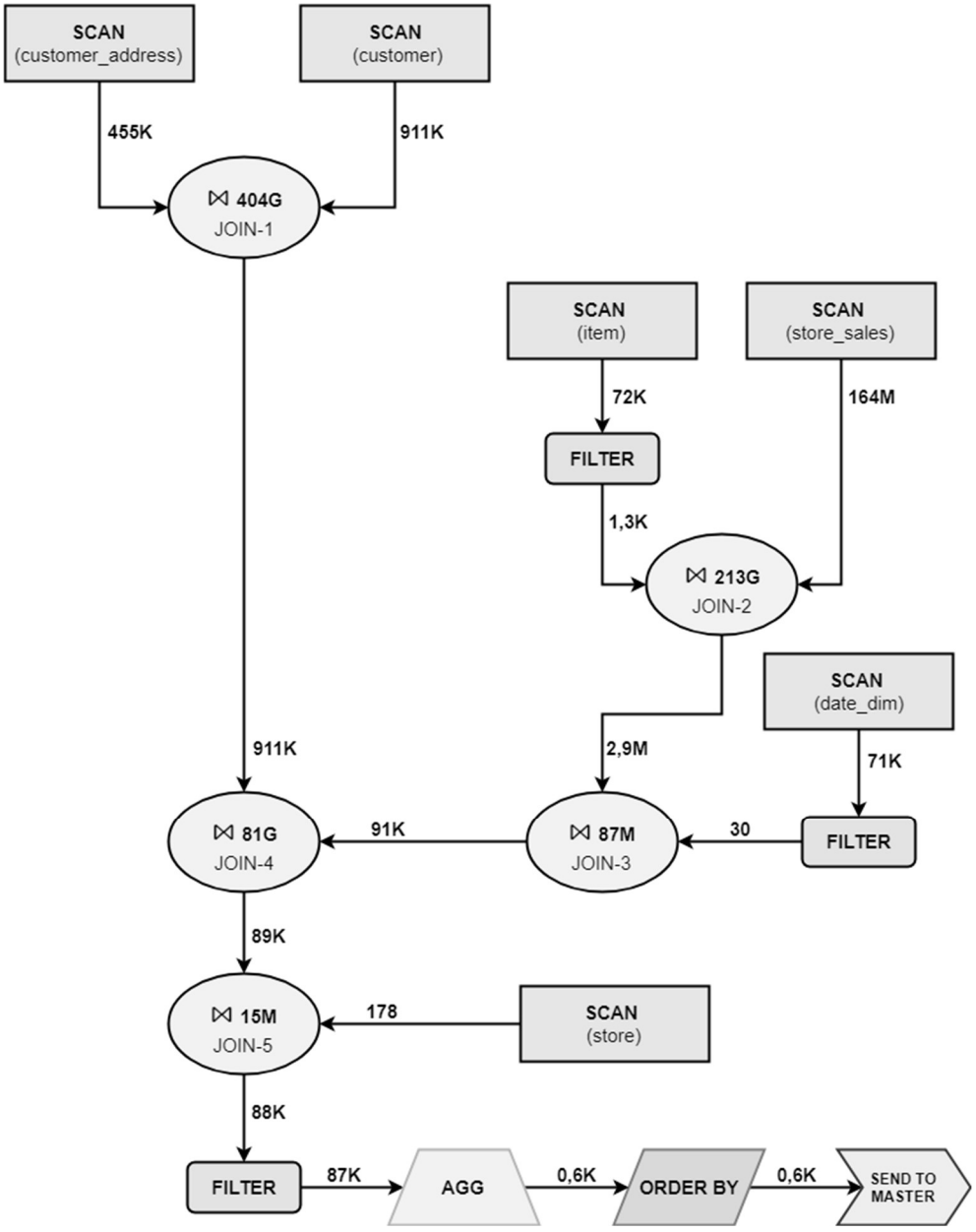

Figura 5.23: Plano de execução da Consulta 91

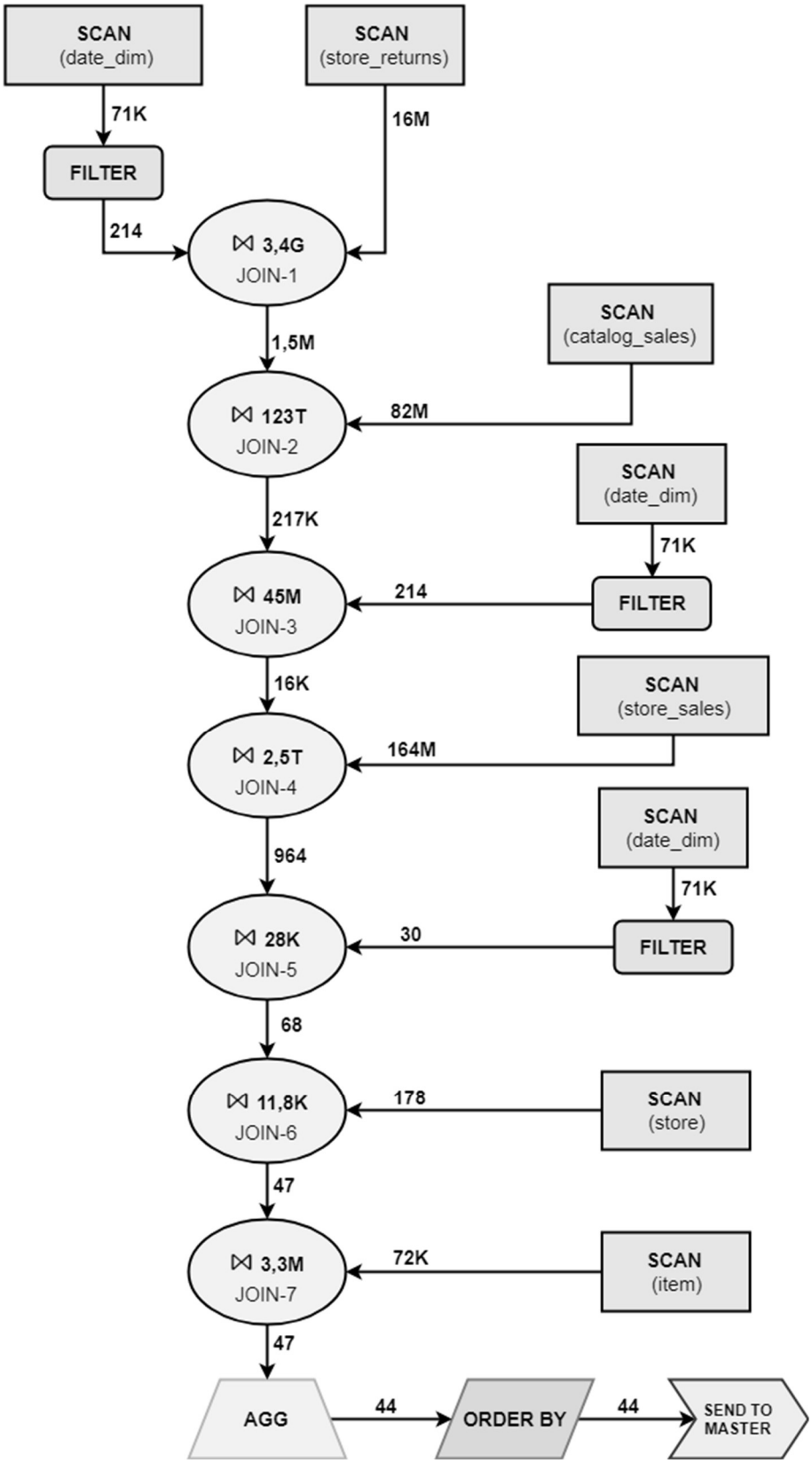

Figura 5.24: Plano de execução da Consulta 96

Tendo em vista as quantidades de tuplas de cada tabela apresentadas na Tabela 5.8 e também os planos de execução apresentados nesta seção, calculamos e tabulamos as cardinalidades e resultados de cada operação destes planos para todas as bases de dados apresentadas na Tabela 5.7. Estes valores são apresentados no ANEXO B.

5.5.4 SCRIPT DE EXECUÇÃO

Para otimizar as implantações de cada cenário utilizado neste experimento no MyriaX para execução da carga de trabalho, utilizamos um *script* que, dada a quantidade de máquinas alocadas, gera vários cenários de distribuição e alocação de *data nodes* e *worker nodes* em máquinas do *cluster* (Tabela 5.6). Tendo em vista o uso de bases de dados diferentes (Tabela 5.7) para cada quantidade de máquinas (2, 4 e 8), submetemos execuções independentes deste *script* ao *cluster* para cada quantidade de máquinas com direcionamento para a base de dados a ser utilizada, totalizando 3 submissões.

Este *script* realiza 5 rodadas de execução da carga de trabalho selecionada do TPC-DS para cada cenário gerado (Tabela 5.6). Cada rodada é executada para todos os cenários gerados antes de iniciar a rodada seguinte, ao invés de todas as 5 rodadas serem executadas de forma sequencial para cada cenário. Com esta lógica, evita-se que um cenário seja totalmente prejudicado por alguma atividade de manutenção do sistema operacional que fuja ao nosso controle durante a execução da carga de trabalho. O fluxograma de execução deste *script* é apresentado na Figura 4.4.

Durante as submissões da carga de trabalho, o *script* captura o tempo que cada consulta integrante da carga de trabalho leva para ser executar. O tempo de execução de uma carga de trabalho é calculado a partir da soma das médias dos tempos de todas as consultas. Ao fim da execução de todas as rodadas para todos os cenários, o *script* calcula a média (por consulta e carga de trabalho) das 5 rodadas de cada cenário, eliminando as rodadas com maior e menor tempo, para cada cenário. Os resultados obtidos são apresentados e avaliados na próxima seção.

5.6 RESULTADOS E AVALIAÇÕES

Após o fim da execução do Experimento II no *cluster*, que contou com submissões do *script* de execução (5.5.4), tabulamos os resultados para avaliação. Estas submissões contaram com experimentos com cenários para 2 máquinas sob base de dados de 20GB, 4 máquinas sob bases de dados de 20GB e 40GB e 8 máquinas sob bases de dados de 40GB e 60GB. O cálculo do percentual de aceleração/desaceleração de cenários *Avaliação* em relação a cenários *Baseline* segue a mesma fórmula utilizada no Experimento I. Nesta seção, discutimos os

resultados para cada consulta individual agrupadas por desempenho com aceleração e desaceleração (Seção 5.6.1), comparamos consultas com características semelhantes (Seção 5.6.2) e avaliamos os resultados para a carga de trabalho (Seção 5.6.3).

5.6.1 CONSULTAS

Nesta seção são apresentados e avaliados os resultados para cada consulta individual referentes às execuções em cada quantidade de máquinas. Agrupamos as consultas integrantes da carga de trabalho em dois grupos: Grupo 1 como sendo consultas que apresentaram aceleração (5.6.1.1) e Grupo 2 como sendo consultas que apresentaram desaceleração e/ou aceleração pouco significativa (5.6.1.2) em cenários *Avaliação*.

5.6.1.1 GRUPO 1: CONSULTAS COM ACELERAÇÃO

Esta seção apresenta discussão sobre os resultados de consultas que apresentaram aceleração em cenários *Avaliação* para ao menos uma quantidade de máquinas. Dito isto, são discutidos os resultados da Consulta 7 (Figura 5.25), Consulta 15 (Figura 5.26), Consulta 25 (Figura 5.27), Consulta 26 (Figura 5.28), Consulta 29 (Figura 5.29), Consulta 48 (Figura 5.30), Consulta 91 (Figura 5.31) e Consulta 96 (Figura 5.32).

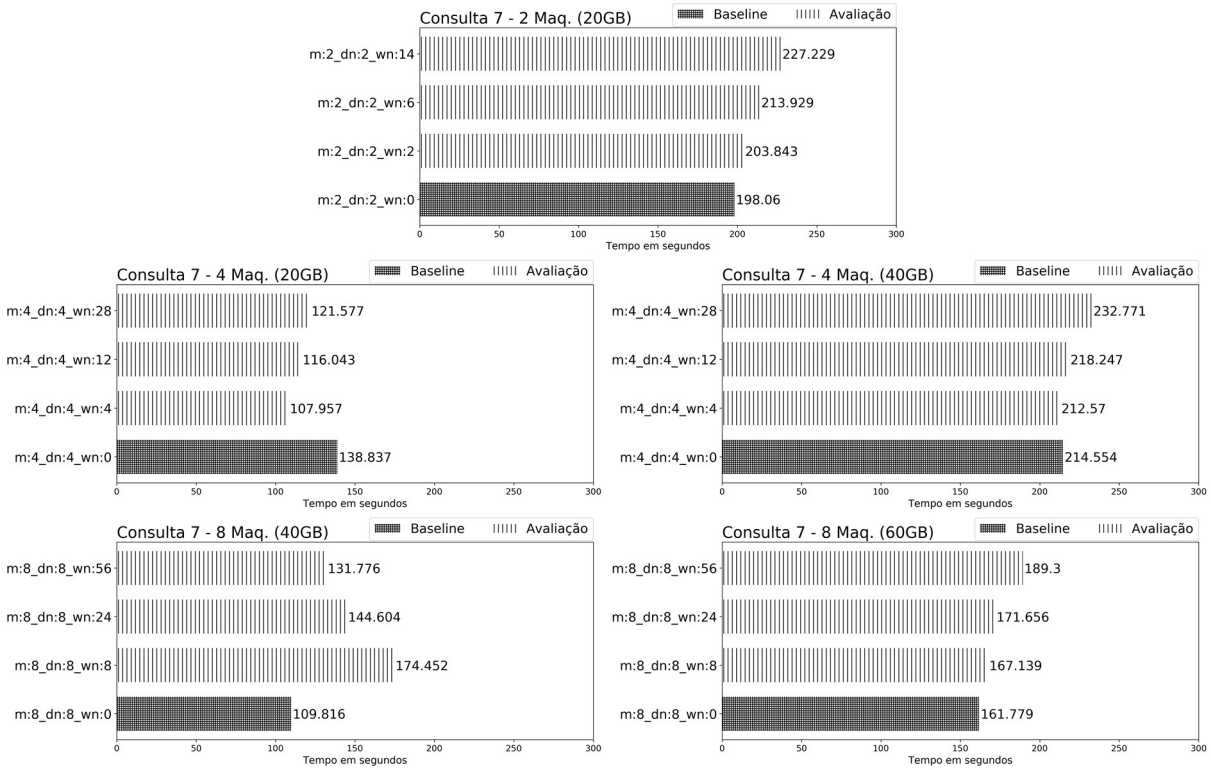

Figura 5.25: Resultados para a Consulta 7

Consulta 7: esta consulta apresentou desaceleração gradativa, para todas as quantidades de máquinas com exceção dos cenários com 4 máquinas e base de dados 20GB, com o acréscimo de *worker nodes* conforme mostra a Figura 5.25. Para cenários com 4 máquinas (20GB), é possível notar aceleração de até 28,6% em cenários *Avaliação* em relação ao cenário *Baseline,* seguida de desaceleração com o acréscimo de *worker nodes*. O pior caso para as demais quantidades de máquinas foram cenários *Avaliação* que utilizam a quantidade máxima de nós totais quando comparado com o cenário *Baseline*, com exceção dos cenários com 8 máquinas e base de dados 40GB. Sendo assim, a abordagem de *data* e *worker nodes* apresentou desaceleração de até 14,73% utilizando cenários com 2 máquinas, 8,49% utilizando cenários com 4 máquinas (40GB) e 17,01% utilizando cenários com 8 máquinas (60GB). Para cenários com 8 máquinas (40GB), é possível notar aceleração em cenários *Avaliação* com o acréscimo de *worker nodes*, mas desaceleração em relação ao cenário *Baseline*.

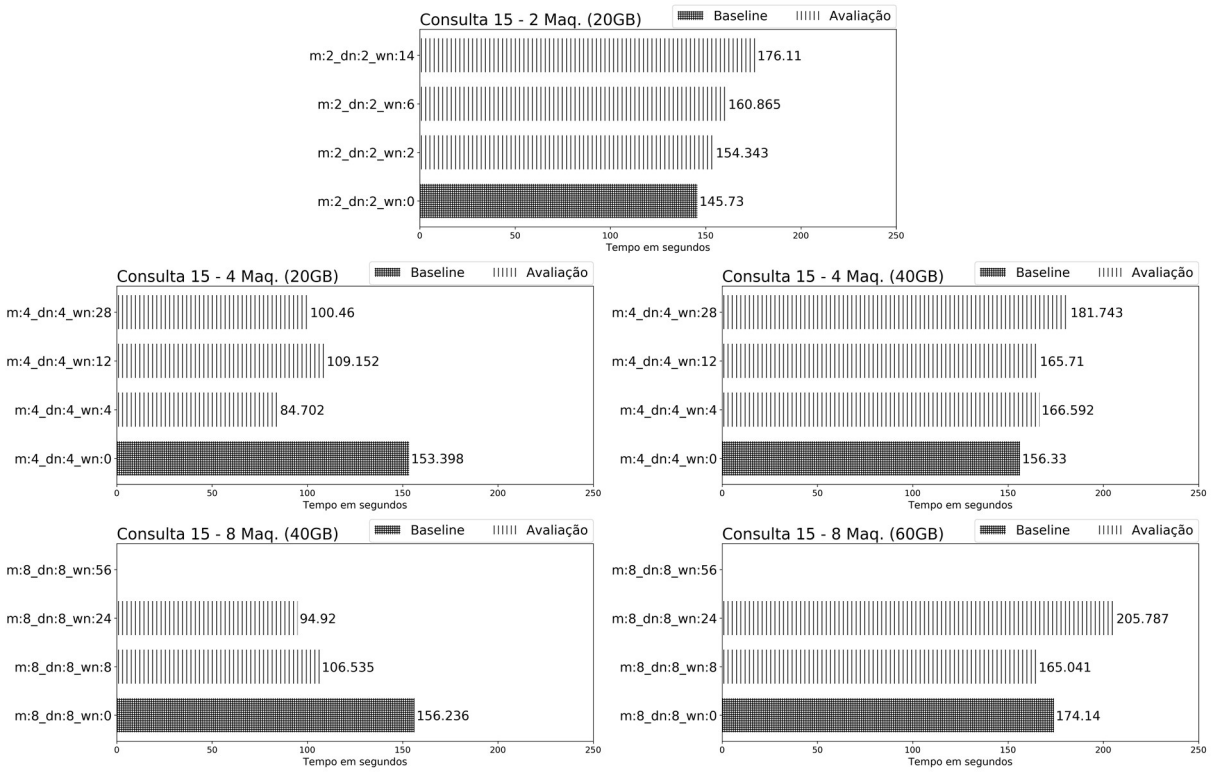

Figura 5.26: Resultados para a Consulta 15

Consulta 15: esta consulta também apresentou desaceleração gradativa, para todas as quantidades de máquinas com exceção dos cenários com 4 máquinas (20GB) e 8 máquinas e (40GB), com o acréscimo de *worker nodes* conforme mostra a Figura 5.26. Os cenários com 4 máquinas (20GB) e 8 máquinas (40GB) apresentaram aceleração. Cenários com 4 máquinas (20GB) apresentaram aceleração de até 81,1% seguida de desaceleração com o acréscimo de *worker nodes*. Cenários com 8 máquinas (40GB) apresentaram aceleração de até 64,6% com

acréscimo de *worker nodes*. Por outro lado, os piores casos em quantidades de máquinas com desaceleração gradativa foram cenários *Avaliação* que utilizam a quantidade máxima de nós totais quando comparado com o cenário *Baseline*. Os resultados para cenários com 8 máquinas não incluem valores para o cenário com 64 nós totais (*m:8_dn:8_wn:56*), pois esta consulta apresentou problemas durante a execução sob este cenário devido à limitação de memória principal das máquinas do *cluster*. Sendo assim, a abordagem de *data* e *worker nodes* apresentou desaceleração de até 20,85% utilizando cenários com 2 máquinas (20GB) (comparação entre os cenários *Avaliação m:2_dn:2_wn:14* e *Baseline m:2_dn:2_wn:0*), 16,26% utilizando cenários com 4 máquinas (comparação entre os cenários *Avaliação m:4_dn:4_wn:28* e *Baseline m:4_dn:4_wn:0* sob base de dados 40GB) e 18,17% utilizando cenários com 8 máquinas (comparação entre os cenários *Avaliação m:8_dn:8_wn:24* e *Baseline m:8_dn:8_wn:0* sob base de dados 60GB).

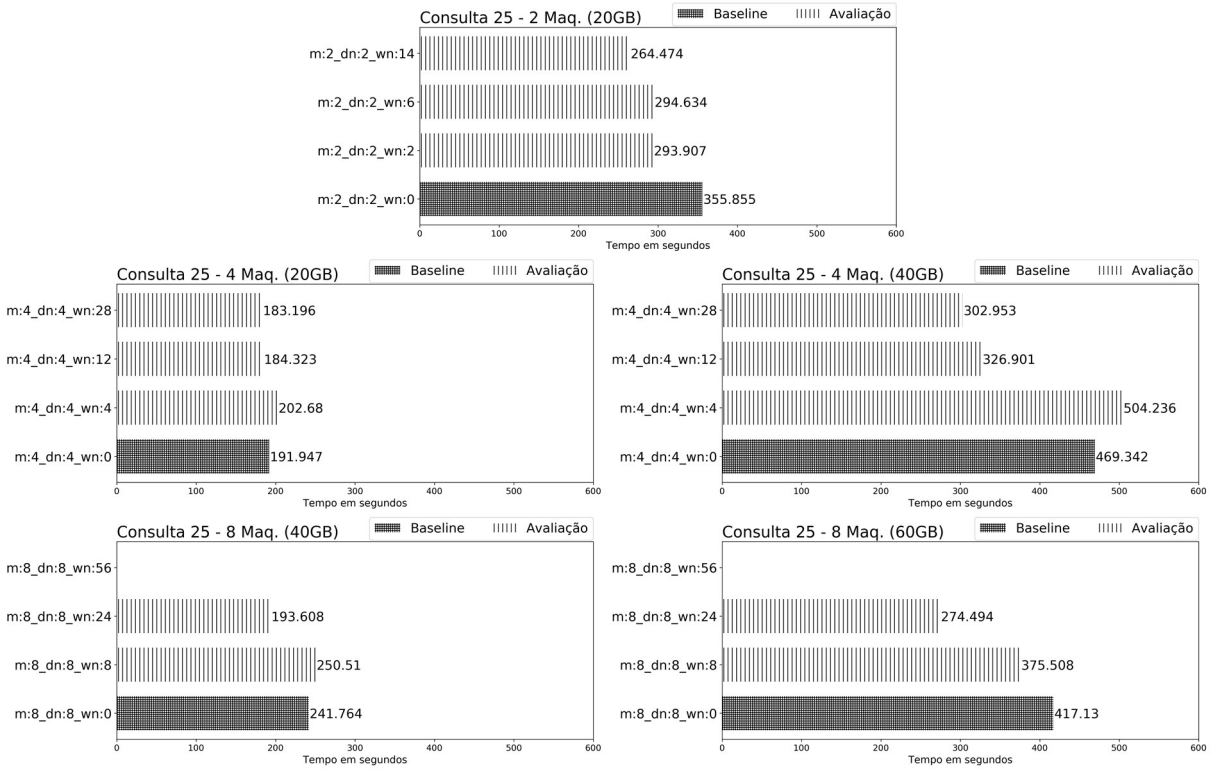

Figura 5.27: Resultados para a Consulta 25

Consulta 25: esta consulta apresentou aceleração gradativa, com o acréscimo de *worker nodes* conforme apresenta a Figura 5.27. Os resultados para cenários com 4 máquinas apresentaram desaceleração com o primeiro cenário *Avaliação* (*m:4_dn:4_wn:4*), mas aceleração para os demais cenários *Avaliação* quando comparados com o cenário Baseline. Os resultados para cenários com 8 máquinas não incluem valores para o cenário com 64 nós totais (*m:8_dn:8_wn:56*), pois esta consulta apresentou problemas durante a execução sob este

cenário devido à limitação de memória principal das máquinas do *cluster*, assim como a Consulta 15. O melhor caso em cada quantidade de máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais quando comparado com o cenário *Baseline*. Sendo assim, a abordagem de *data* e *worker nodes* apresentou aceleração de até 34,55% utilizando cenários com 2 máquinas (20GB) (comparação entre os cenários *Avaliação m:2_dn:2_wn:14* e *Baseline m:2_dn:2_wn:0*); 4,78% (20GB) e 54,92% (40GB) utilizando cenários com 4 máquinas (comparação entre os cenários *Avaliação m:4_dn:4_wn:28* e *Baseline m:4_dn:4_wn:0*); e 24,87% (40GB) 51,96% (60GB) utilizando cenários com 8 máquinas (comparação entre os cenários *Avaliação m:8_dn:8_wn:24* e *Baseline m:8_dn:8_wn:0*). É possível notar maior aceleração no processamento de carga de trabalho para a mesma quantidade de máquina com base de dados de maior fator de geração.

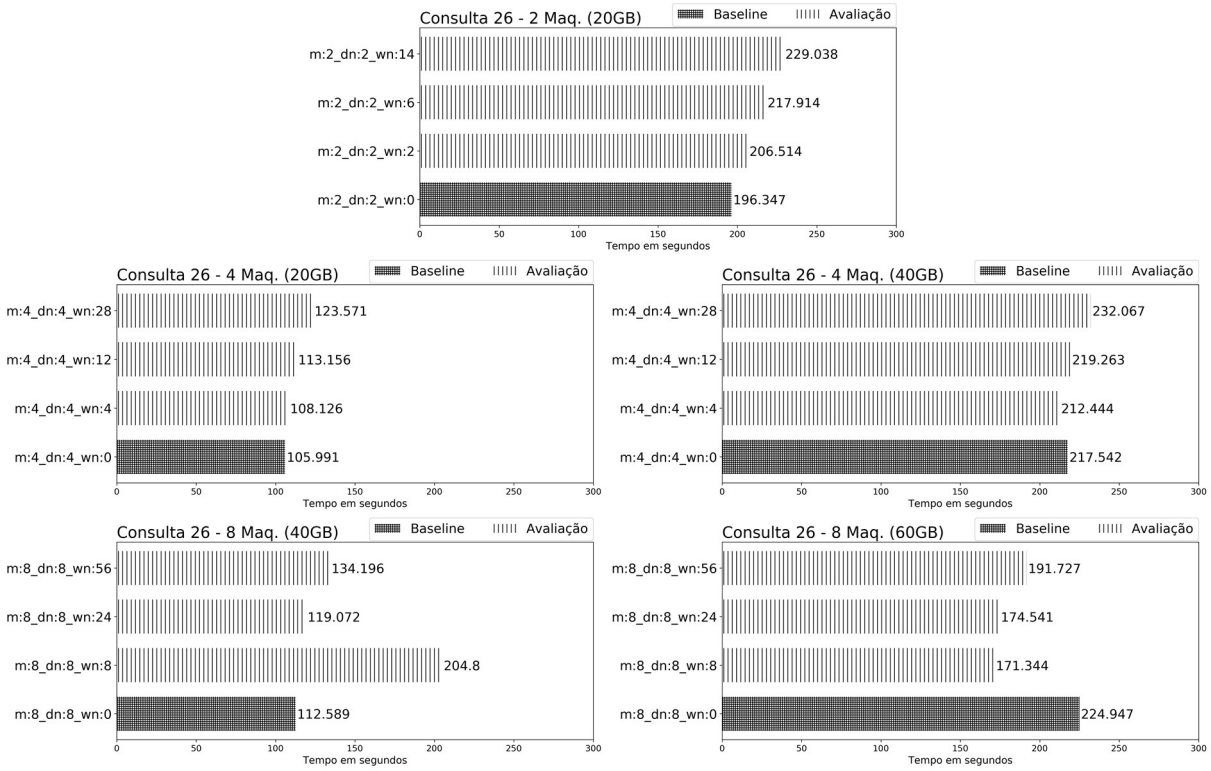

Figura 5.28: Resultados para a Consulta 26

Consulta 26: para esta consulta, conforme apresenta a Figura 5.28, os resultados para cenários com 2 máquinas (20GB) apresentaram desaceleração gradativa com o acréscimo de *worker nodes*. O pior caso para cenários com 2 máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais (*m:2_dn:2_wn:14*) quando comparado com o cenário *Baseline* (*m:2_dn:2_wn:0*), desacelerando 16,65%. Porém, os resultados para cenários com 4 e 8 máquinas (40GB e 60GB respectivamente) apresentaram aceleração e posterior desaceleração de acordo com o acréscimo de *worker nodes*. Para cenários com 4 máquinas (40GB), o cenário

Avaliação m:4_dn:4_wn:4 apresentou o melhor resultado. Este cenário acelerou 2,40% em relação ao cenário *Baseline m:4_dn:4_wn:0*. Mas o cenário *Avaliação m:4_dn:4_wn:28* apresentou desaceleração de 6,68% em relação ao cenário *Baseline m:4_dn:4_wn:0* e 9,24% em relação ao cenário *Avaliação m:4_dn:4_wn:4*. Para cenários com 8 máquinas (60GB) o cenário *Avaliação m:8_dn:8_wn:8* apresentou o melhor resultado, acelerando 31,28% em relação ao cenário *Baseline m:8_dn:8_wn:0*. Mas o cenário *Avaliação m:8_dn:8_wn:56* apresentou aceleração inferior de 17,33% em relação ao cenário *Baseline m:8_dn:8_wn:0* e desaceleração de 11,63% em relação ao cenário *Avaliação m:8_dn:8_wn:8*. Para esta consulta é possível notar que o acréscimo de *worker nodes* no processamento de cargas de trabalho em 4 (40GB) e 8 (60GB) máquinas causou aceleração seguida de desaceleração em certo ponto. Para os demais casos, o acréscimo de *worker nodes* causou desaceleração gradativa.

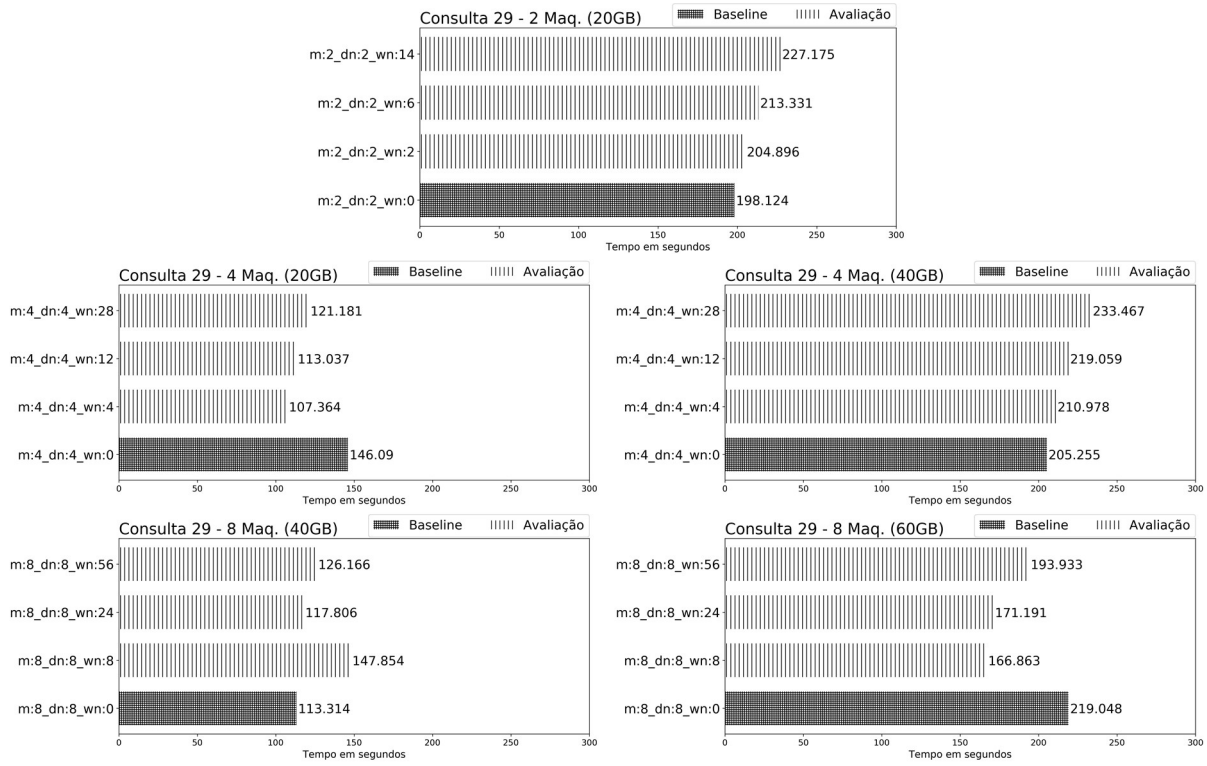

Figura 5.29: Resultados para a Consulta 29

Consulta 29: para esta consulta, conforme apresenta a Figura 5.29, os resultados com cenários de 2 máquinas (20GB), 4 máquinas (40GB) e 8 máquinas (40GB) apresentaram desaceleração gradativa com o acréscimo de *worker nodes*. O pior caso para estas quantidades de máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais quando comparado com o cenário *Baseline*, desacelerando 14,66% para cenários com 2 máquinas (20GB), 13,74% para cenários com 4 máquinas (40GB) e 30,48% para cenários com 8 máquinas (40GB). Porém, os resultados para cenários com 4 e 8 máquinas (20GB e 60GB

respectivamente) apresentaram aceleração e posterior desaceleração de acordo com o acréscimo de *worker nodes*. Considerando cenários com 8 máquinas (60GB), o cenário *Avaliação m:8_dn:8_wn:8* apresentou o melhor resultado, acelerando 31,27% em relação ao cenário *Baseline m:8_dn:8_wn:0*. Mas o cenário *Avaliação m:8_dn:8_wn:56* apresentou aceleração inferior de 12,95% em relação ao cenário *Baseline m:8_dn:8_wn:0* e desaceleração de 16,22% em relação ao cenário *Avaliação m:8_dn:8_wn:8*. Os demais casos apresentaram desaceleração gradativa com o acréscimo de *worker nodes*.

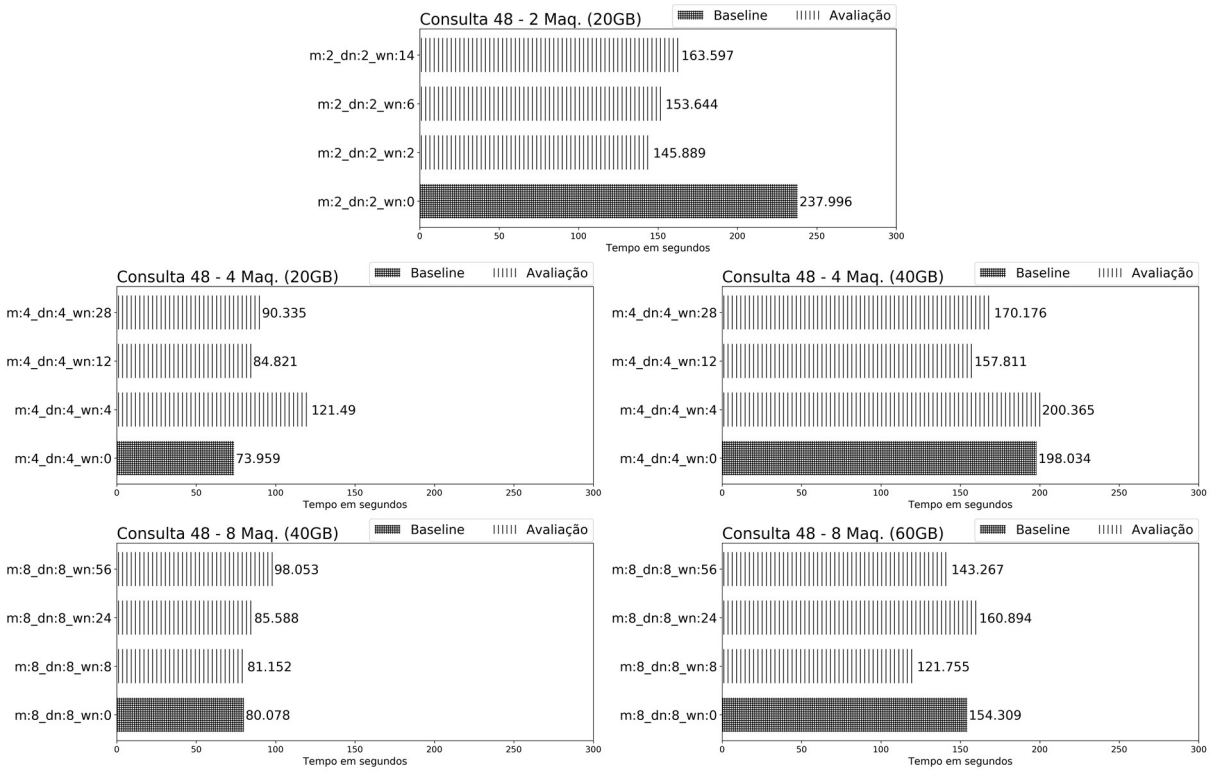

Figura 5.30: Resultados para a Consulta 48

Consulta 48: assim como os resultados para cenários com 8 máquinas das Consultas 26 e 29, esta consulta apresentou aceleração e posterior desaceleração de acordo com o acréscimo de *worker nodes.* Para cenários com 2 máquinas (20GB), o cenário *Avaliação m:2_dn:2_wn:2* apresentou o melhor resultado, acelerando 63,13% em relação ao cenário *Baseline m:2_dn:2_wn:0*. Mas o cenário *Avaliação m:2_dn:2_wn:14* apresentou aceleração inferior de 45,48% em relação ao cenário *Baseline m:2_dn:2_wn:0* e desaceleração de 12,12% em relação ao cenário *Avaliação m:2_dn:2_wn:2*. Para cenários com 4 máquinas (40GB), o cenário *Avaliação m:4_dn:4_wn:12* apresentou o melhor resultado, acelerando 25,49% em relação ao cenário *Baseline m:4_dn:4_wn:0*. Mas o cenário *Avaliação m:4_dn:4_wn:28* apresentou aceleração inferior de 16,37% em relação ao cenário *Baseline m:4_dn:4_wn:0* e desaceleração de 7,84% em relação ao cenário *Avaliação m:4_dn:4_wn:12*. Para cenários com 8 máquinas

(40GB), cenários *Avaliação* apresentaram desaceleração de até 22,44% com acréscimo de *worker nodes*. Para cenários com 8 máquinas (60GB), o cenário *Avaliação m:8_dn:8_wn:8* apresentou o melhor resultado, acelerando 26,74% em relação ao cenário *Baseline m:8_dn:8_wn:0*. Mas os cenários *Avaliação m:8_dn:8_wn:24* e *m:8_dn:8_wn:56* apresentaram desaceleração de 4,27% e aceleração de 7,16%, respectivamente, em relação ao cenário *Baseline m:8_dn:8_wn:0*; e desaceleração de 32,15% e 17,67%, respectivamente, em relação ao cenário *Avaliação m:8_dn:8_wn:8*. Os demais casos apresentaram desaceleração gradativa com o acréscimo de *worker nodes*.

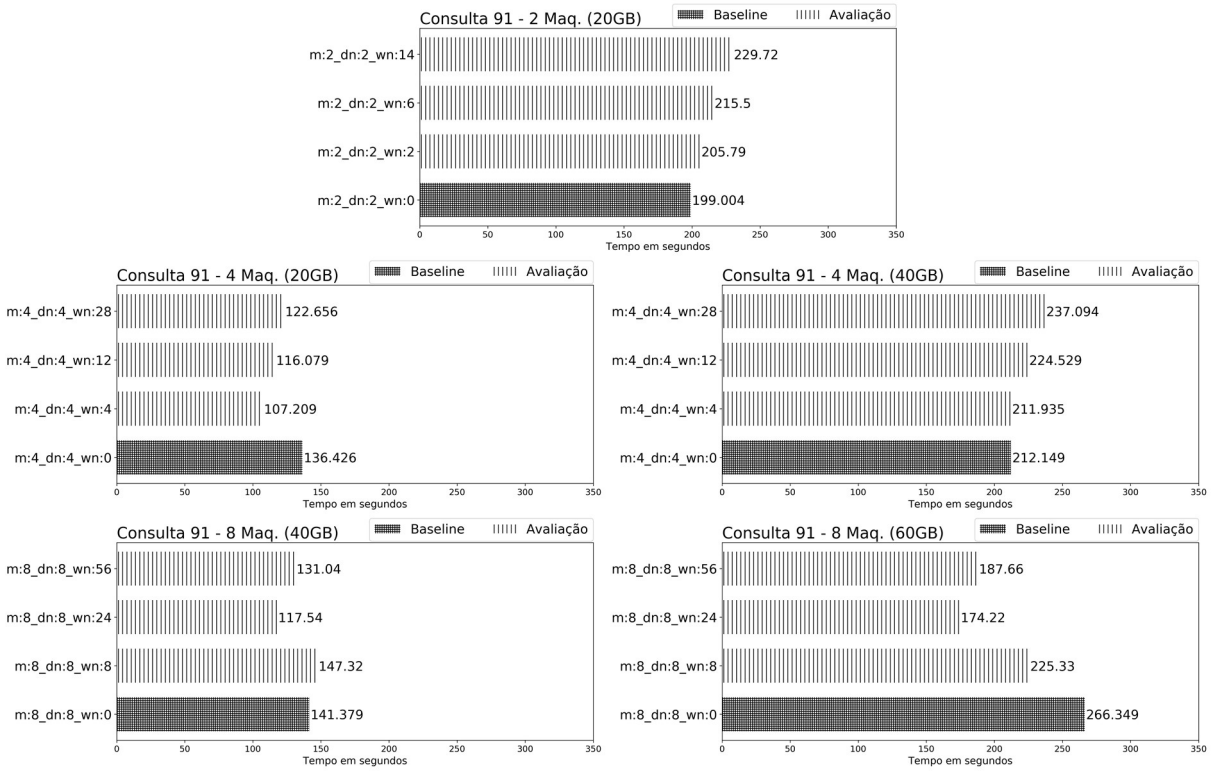

Figura 5.31: Resultados para a Consulta 91

Consulta 91: assim como a Consulta 29, conforme apresenta a Figura 5.31, os resultados para esta consulta com cenários de 2 máquinas (20GB) e 4 máquinas (40GB) apresentaram desaceleração gradativa com o acréscimo de *worker nodes*. O pior caso para estas quantidades de máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais quando comparado com o cenário *Baseline*, desacelerando 15,43% para cenários com 2 máquinas e 11,76% para cenários com 4 máquinas (40GB). Porém, os resultados para cenários com 4 máquinas (20GB) 8 máquinas apresentaram aceleração e posterior desaceleração de acordo com o acréscimo de *worker nodes*. O cenário *Avaliação m:8_dn:8_wn:24* apresentou o melhor resultado, acelerando 20,28% (40GB) 52,88% (60GB) em relação ao cenário *Baseline m:8_dn:8_wn:0*. Mas o cenário *Avaliação m:8_dn:8_wn:56* apresentou aceleração inferior de

7,89% (40GB) 41,93% (60GB) em relação ao cenário *Baseline m:8_dn:8_wn:0* e desaceleração de 11,49% (40GB) e 7,71% (60GB) em relação ao cenário *Avaliação m:8_dn:8_wn:24*. Para esta consulta é possível notar que o acréscimo de *worker nodes* no processamento de cargas de trabalho em 4 máquinas (20GB) e 8 máquinas causou aceleração seguida de desaceleração em certo ponto.

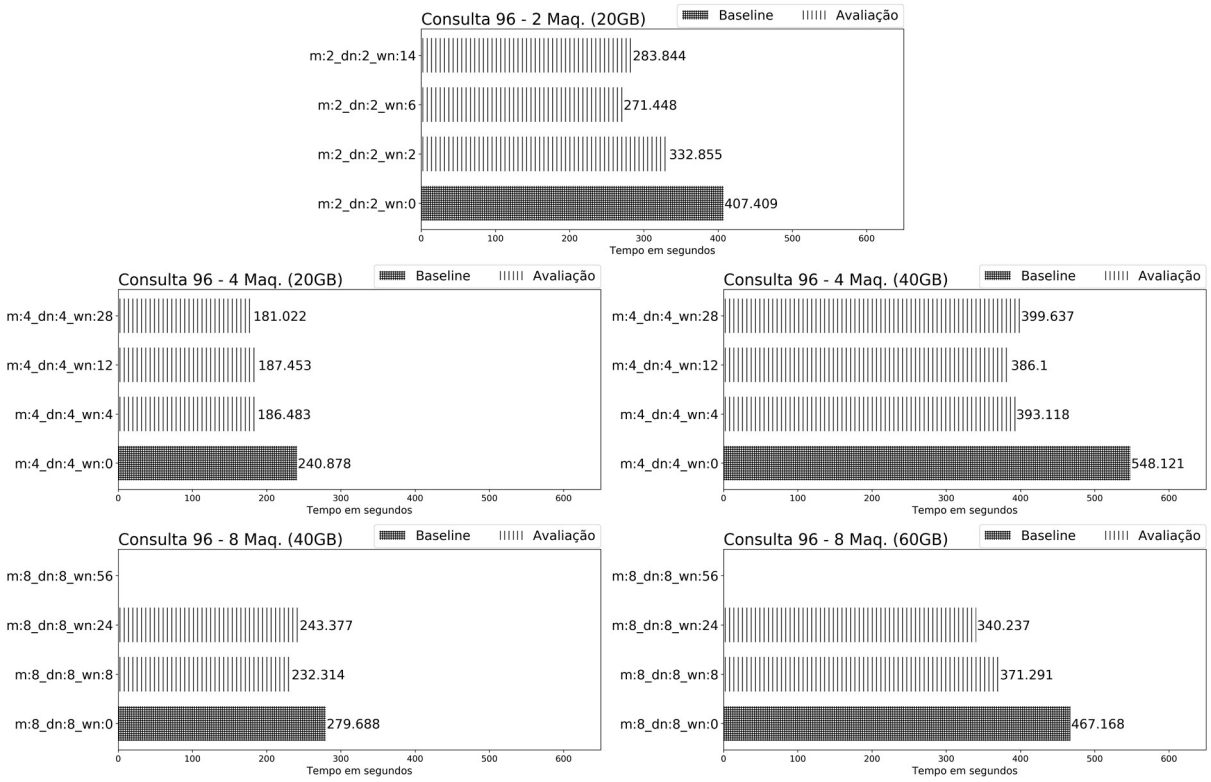

Figura 5.32: Resultados para a Consulta 96

Consulta 96: esta consulta também apresentou resultados com aceleração e posterior desaceleração com o acréscimo de *worker nodes*, conforme apresenta a Figura 5.32. Para cenários com 2 máquinas (20GB), o cenário *Avaliação m:2_dn:2_wn:6* apresentou o melhor resultado, acelerando 50,09% em relação ao cenário *Baseline m:2_dn:2_wn:0*. Mas o cenário *Avaliação m:2_dn:2_wn:14* apresentou aceleração inferior de 43,53% em relação ao cenário *Baseline m:2_dn:2_wn:0* e desaceleração de 4,57% em relação ao cenário *Avaliação m:2_dn:2_wn:6*. Para cenários com 4 máquinas (20GB), cenários *Avaliação* alcançaram aceleração de até 33,07%. Em cenários com 4 máquinas (40GB), o cenário *Avaliação m:4_dn:4_wn:12* apresentou o melhor resultado com aceleração de até 41,96% em relação ao cenário *Baseline m:4_dn:4_wn:0*. Mas o cenário *m:4_dn:4_wn:28* apresentou aceleração inferior de 37,15% em relação ao cenário *Baseline m:4_dn:4_wn:0* e desaceleração de 3,51% em relação ao cenário *Avaliação m:4_dn:4_wn:12*. Os resultados para cenários com 8 máquinas não incluem valores para o cenário com 64 nós totais (*m:8_dn:8_wn:56*), pois esta consulta

também apresentou problemas durante a execução sob este cenário devido à limitação de memória principal das máquinas do *cluster*, assim como as Consultas 15 e 25. Neste experimento, o cenário *Avaliação m:8_dn:8_wn:24* (60GB) apresentou o melhor resultado, acelerando 37,31% em relação ao cenário *Baseline m:8_dn:8_wn:0*.

Este grupo de consultas apresentou considerável aceleração em cenários *Avaliação* durante a execução em ao menos uma quantidade de máquinas, e foi possível notar uma característica predominante: o acréscimo de *worker nodes* causa aceleração até certo ponto e depois desaceleração. A Figura 5.33 apresenta os principais resultados para aceleração e desaceleração das consultas deste grupo. Foi possível notar também que algumas consultas apresentaram desempenhos diferentes para a mesma quantidade de máquinas e base de dados diferentes, tais como as Consultas 7 e 15. As Consultas 25 e 96 não apresentaram resultado para o cenário com 8 máquinas *m:8_dn:8_wn:54* devido a problemas durante a execução, mas acreditamos que há uma grande probabilidade deste cenário apresentar desaceleração em relação ao melhor caso desta quantidade de máquinas, seguindo a característica predominante das demais consultas.

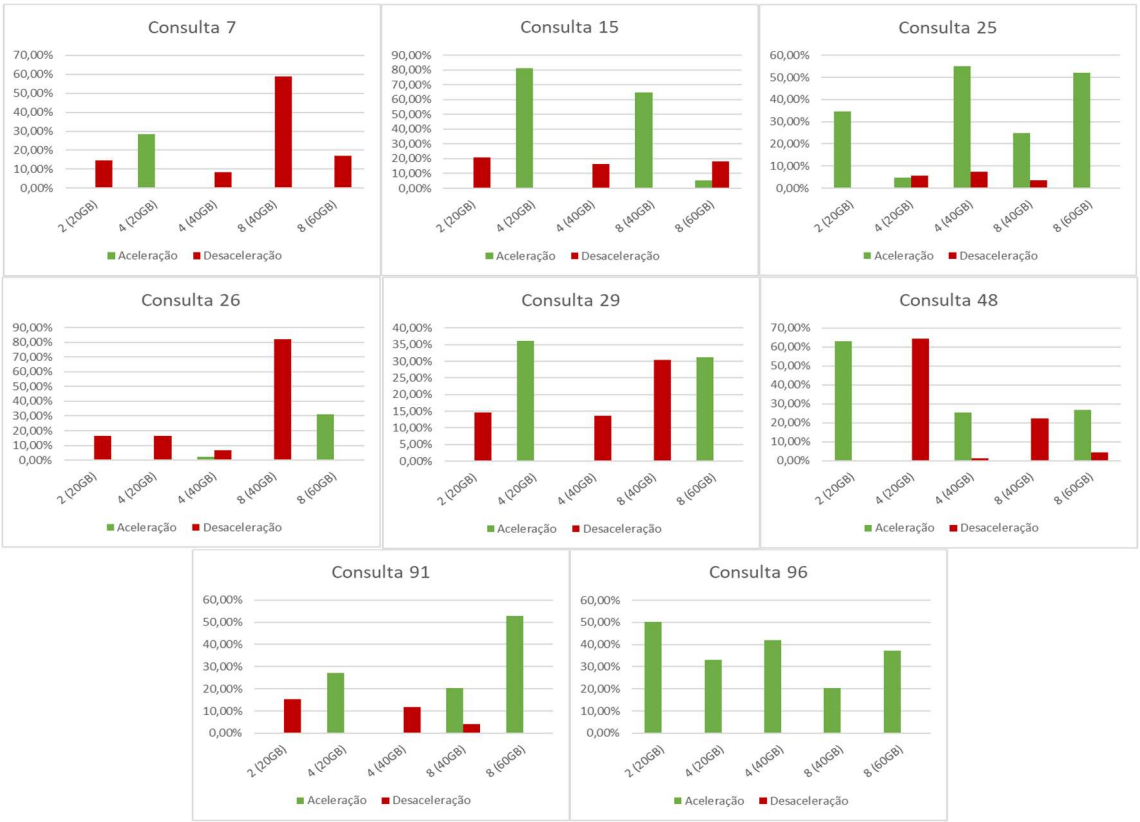

Figura 5.33: Resultados para Consultas do Grupo 1

5.6.1.2 GRUPO 2: CONSULTAS COM DESACELERAÇÃO

Esta seção apresenta discussão sobre os resultados de consultas que apresentaram desaceleração e/ou aceleração pouco significativa em cenários *Avaliação*. Dito isto, são discutidos os resultados da Consulta 17 (Figura 5.34), Consulta 19 (Figura 5.35) e Consulta 27 (Figura 5.36).

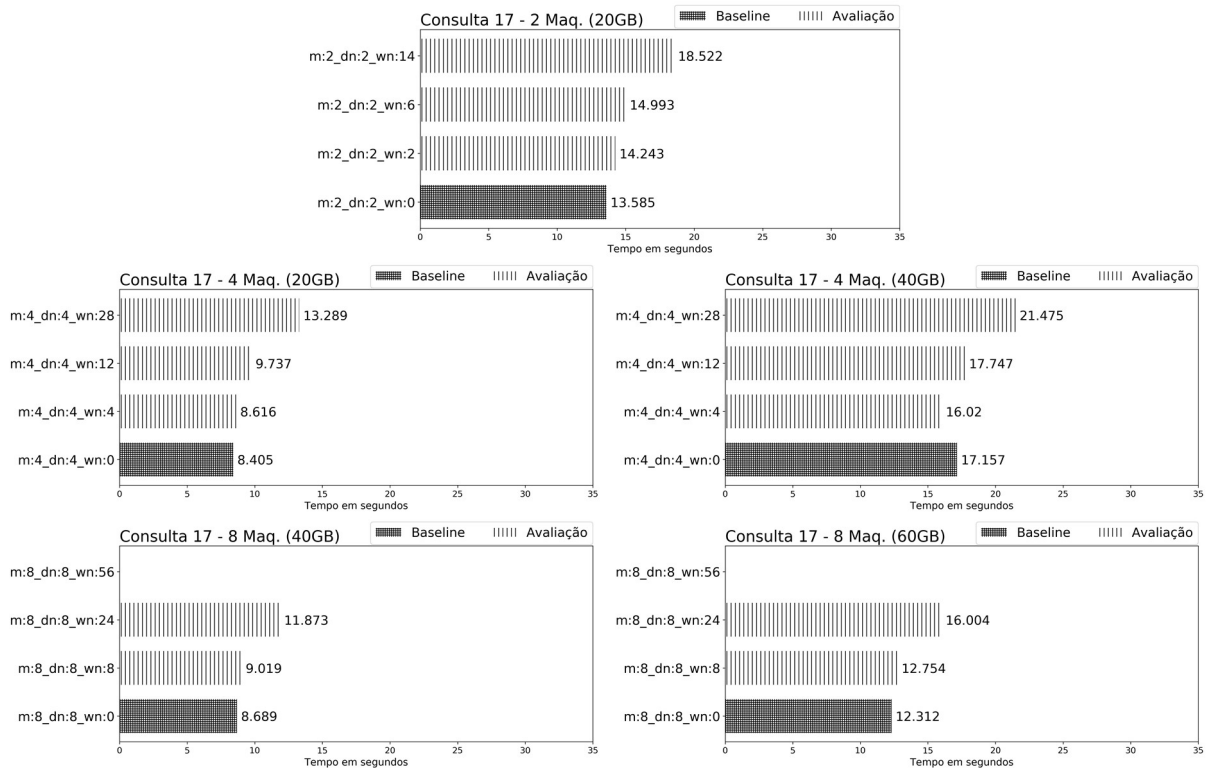

Figura 5.34: Resultados para a Consulta 17

Consulta 17: esta consulta também apresentou desaceleração gradativa ou aceleração desprezível, para todas as quantidades de máquinas, com o acréscimo de *worker nodes* conforme mostra a Figura 5.34. É possível notar que os cenários com 4 máquinas (40GB) apresentaram desprezível aceleração de 7,1% (comparação entre os cenários *Avaliação m:4_dn:4_wn:4* e *Baseline m:4_dn:4_wn:0*) seguida de desaceleração em cenários *Avaliação*. O pior caso em cada quantidade de máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais quando comparado com o cenário *Baseline*. Os resultados para cenários com 8 máquinas não incluem valores para o cenário com 64 nós totais (*m:8_dn:8_wn:56*), pois esta consulta também apresentou problemas durante a execução sob este cenário devido à limitação de memória principal das máquinas do *cluster*, assim como as Consultas 15, 25 e 96. Sendo assim, a abordagem de *data* e *worker nodes* apresentou desaceleração de até 36,34% utilizando cenários com 2 máquinas (20GB) (comparação entre os cenários *Avaliação*

m:2_dn:2_wn:14 e *Baseline m:2_dn:2_wn:0*), 58,11% (20GB) e 25,17% (40GB) utilizando cenários com 4 máquinas (comparação entre os cenários *Avaliação m:4_dn:4_wn:28* e *Baseline m:4_dn:4_wn:0*) e 36,64% (40GB) e 30% (60GB) utilizando cenários com 8 máquinas (comparação entre os cenários *Avaliação m:8_dn:8_wn:24* e *Baseline m:8_dn:8_wn:0*).

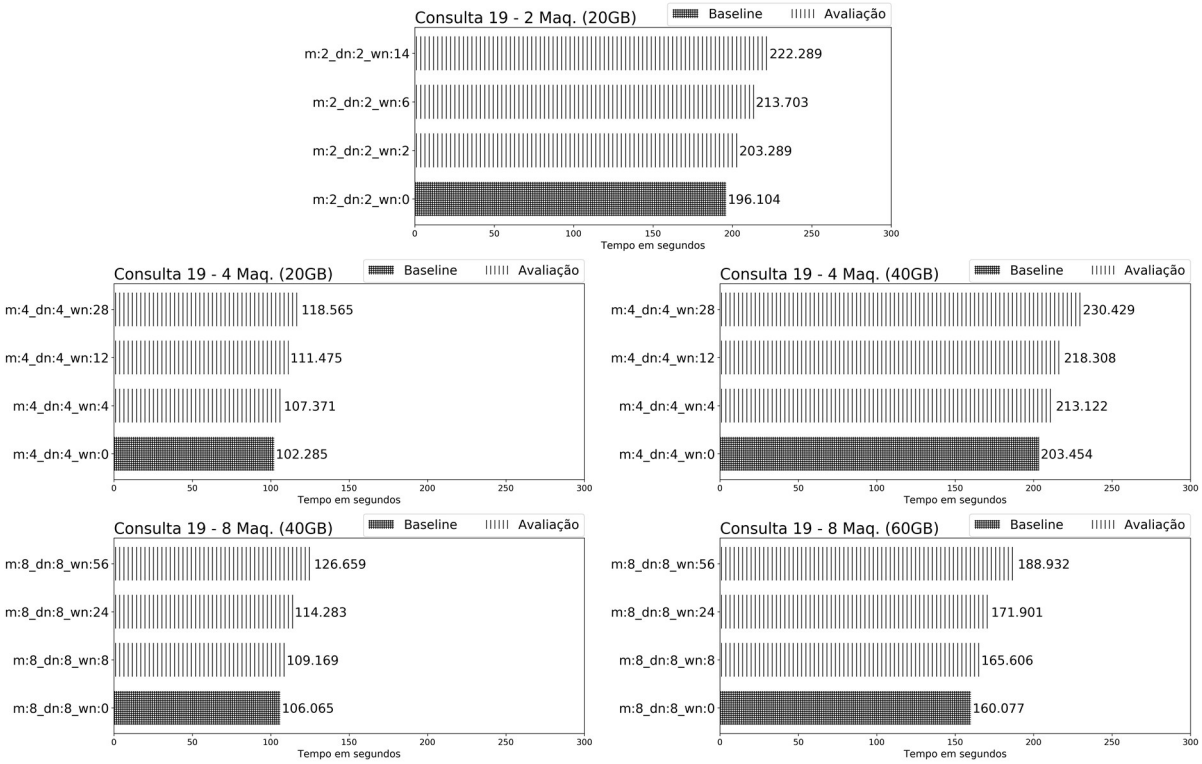

Figura 5.35: Resultados para a Consulta 19

Consulta 19: esta consulta também apresentou desaceleração gradativa, para todas as quantidades de máquinas, com o acréscimo de *worker nodes* conforme apresenta a Figura 5.35. O pior caso em cada quantidade de máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais quando comparado com o cenário *Baseline*. Sendo assim, a abordagem de *data* e *worker nodes* apresentou desaceleração de até 13,35% utilizando cenários com 2 máquinas, 15,92% (20GB) e 13,26% (40GB) utilizando cenários com 4 máquinas e 19,42% (40GB) e18,03% (60GB) utilizando cenários com 8 máquinas.

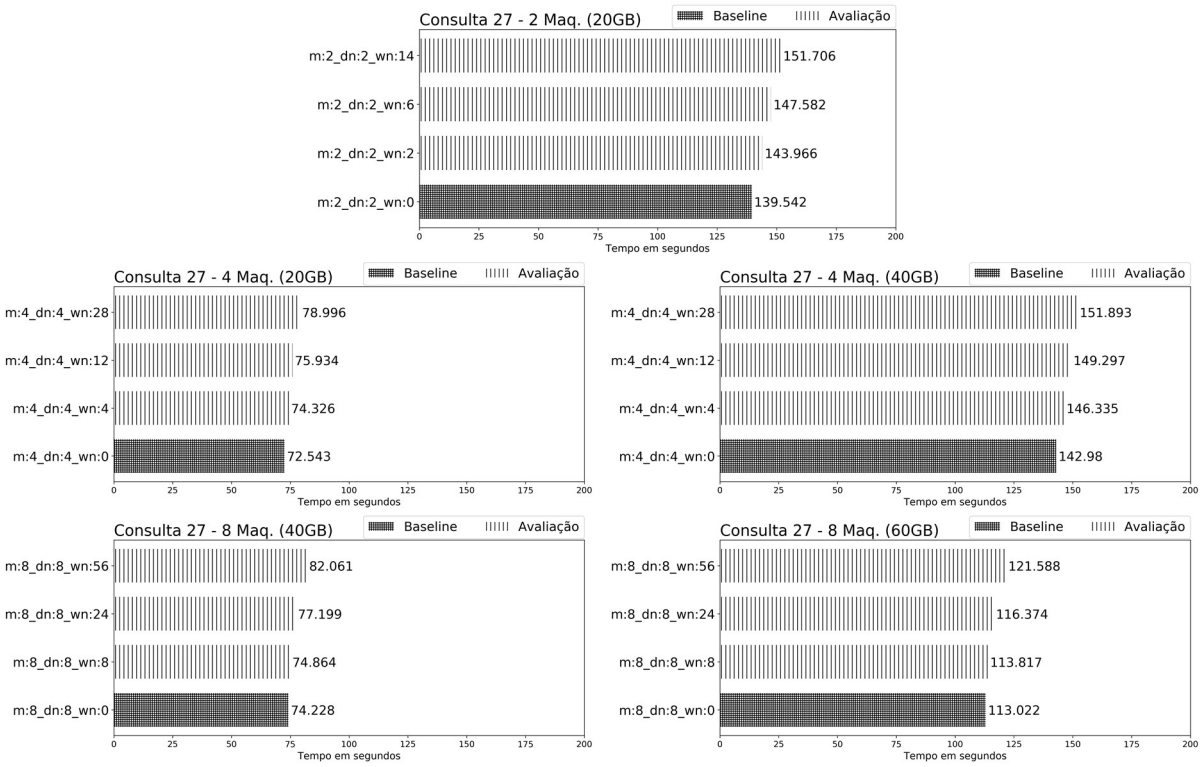

Figura 5.36: Resultados para a Consulta 27

Consulta 27: esta consulta apresentou desaceleração gradativa, para todas as quantidades de máquinas, com o acréscimo de *worker nodes* conforme apresenta a Figura 5.36. O pior caso em cada quantidade de máquinas foi o cenário *Avaliação* que utiliza a quantidade máxima de nós totais quando comparado com o cenário *Baseline*. Sendo assim, a abordagem de *data* e *worker nodes* apresentou desaceleração de até 8,72% utilizando cenários com 2 máquinas, 8,9% (20GB) e 6,23% (40GB) utilizando cenários com 4 máquinas e 10,55% (40GB) e 7,58% (60GB) utilizando cenários com 8 máquinas.

Neste grupo de consultas, foi possível notar que o acréscimo de *worker nodes* piora o desempenho do processamento. A Figura 5.37 apresenta os principais resultados para aceleração e desaceleração das consultas deste grupo. Desconsiderando raros cenários onde há aceleração pouco significante, houve desaceleração em cenários *Avaliação* em todas as consultas em relação aos cenários *Baseline*.

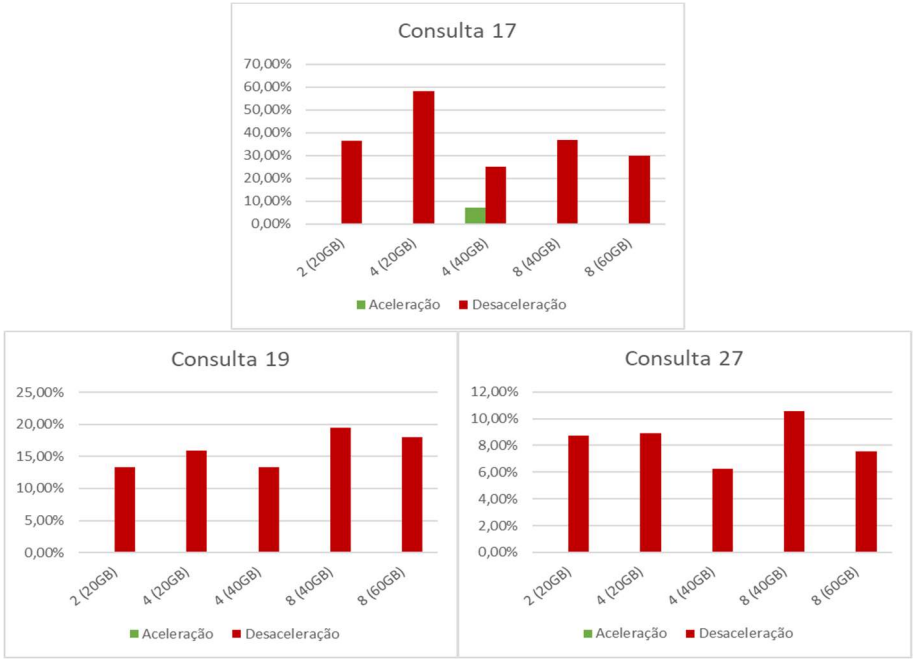

Figura 5.37: Resultados para Consultas do Grupo 2

5.6.2 COMPARATIVO ENTRE CONSULTAS

Esta seção apresenta uma discussão sobre os resultados das consultas apresentados na seção anterior, agrupando-as por características semelhantes. Para essa análise, consideramos os resultados para o experimento com 8 máquinas e base de dados 60GB de cada consulta.

Consultas 91 e 15: as consultas 91 (plano de execução apresentado em Figura 5.23) e 15 (plano de execução apresentado em Figura 5.15) contêm 5 e 6 junções respectivamente. Ambas mantêm o resultado de uma junção em memória enquanto duas junções fora do fluxo sequencial são processadas e executadas, sendo JOIN-2 e JOIN-3 para a Consulta 91 e JOIN-3 e JOIN-4 para a Consulta 15. Diferente da Consulta 91, as duas junções fora do fluxo sequencial da Consulta 15 processam junções complexas, sendo 5,8T e 10T tuplas ($T = 10^{12}$) respectivamente, enquanto que a Consulta 91 processa 213G ($G = 10^{9}$) e 87M ($M = 10^{6}$) tuplas. A Consulta 15 desacelerou 15,38% no pior caso e a Consulta 91 acelerou 52,88% no melhor caso, comparando o cenário *Avaliação m:8_dn:8_wn:24* ao cenário *Baseline m:8_dn:8_wn:0*. Acreditamos que a maior complexidade das junções fora do fluxo sequencial da Consulta 15 influenciaram diretamente na desaceleração desta consulta.

Consultas 17, 25 e 96: as Consultas 17 (plano de execução apresentado em Figura 5.16), 25 (plano de execução apresentado em Figura 5.18) e 96 (plano de execução apresentado em Figura 5.24) contêm 7, 7 e 6 junções respectivamente. Todas as junções seguem um fluxo sequencial sem necessitar armazenar resultado em memória para aguardar resultado de outras

junções. A Consulta 25 apresentou aceleração de 51,96%, a 96 aceleração de 37,31% e a 17 desaceleração de 30%, comparando o cenário *Avaliação m:8_dn:8_wn:24* ao cenário *Baseline m:8_dn:8_wn:0*. Diferente da Consulta 17, as Consultas 25 e 96 apresentam junções internas com alto grau de complexidade, sendo JOIN-2 (24T tuplas) e JOIN-4 (5T tuplas) para Consulta 25 e JOIN-2 (123T tuplas) e JOIN-4 (2,5T tuplas) para Consulta 96. Por outro lado, a Consulta 17 contém a maior junção interna (JOIN-3) com cardinalidade de 67G tuplas. Nota-se que, nestas condições, a abordagem de *data* e *worker nodes* apresentou aceleração com o aumento na complexidade das junções internas.

Consultas 7, 26, 29 e 48: as Consultas 7 (plano de execução apresentado em Figura 5.14), 26 (plano de execução apresentado em Figura 5.19), 29 (plano de execução apresentado em Figura 5.21) e 48 (plano de execução apresentado em Figura 5.22) contêm 4 junções cada. Todas as junções seguem um fluxo sequencial sem necessitar armazenar resultado em memória para aguardar resultado de outras junções. Estas consultas apresentaram desaceleração de 0,7%, aceleração de 31,28%, aceleração de 31,27% e aceleração de 26,74% respectivamente, comparando o cenário *Avaliação m:8_dn:8_wn:8* (melhor cenário para as Consultas 26, 29 e 48). A consulta 7 apresentou desaceleração gradativa com o acréscimo de *worker nodes*. A Consulta 7 é muito semelhante à consulta 26, em que esta realiza leitura de apenas uma tabela diferente e as duas últimas junções contêm cardinalidades menores. A partir de nossas avaliações, não encontramos um motivo concreto para justificar a desaceleração da Consulta 7.

Consultas 19 e 27: as Consultas 19 e 27 também apresentam estruturas bem semelhantes: 4 tabelas e três junções. Ambas apresentaram desaceleração em todos os cenários *Avaliação*: 18,03% e 7,58% no pior caso, respectivamente. Desconsiderando as operações de filtro, a Consulta 19 contém 2,2G tuplas e 7,4M tuplas nas duas últimas junções, enquanto que a Consulta 27 contém cardinalidades de 2T e 1T tuplas. A partir de nossas avaliações, concluímos que neste caso o aumento da complexidade das junções reduziu a desaceleração.

Com base nos planos de execução apresentados na Seção 5.5.3, tabulamos características importantes das consultas integrantes da carga de trabalho, quantidade de junções e cardinalidade da maior junção. Esta tabulação considerou o processamento destas consultas em cenários com 8 máquinas sob a Base 3 (60GB) (Seção 5.5.2) e é apresentada na Tabela 5.9, ordenada do melhor para pior resultado considerando cenários com 8 máquinas (60GB). As consultas destacadas em cinza apresentaram aceleração e as demais apresentaram desaceleração em cenários *Avaliação* em relação aos cenários *Baseline*.

Tabela 5.9: Agrupamento de consultas por características considerando a Base 3 (60GB)

Consulta	Agregação	*Order By*	*Distinct*	*Substring*	# Junções	Maior Junção	Cardinalidade da Maior Junção
91	Sim	Sim	Não	Não	5	JOIN-1	404G
25	Sim	Sim	Não	Não	7	JOIN-2	24T
26	Sim	Não	Não	Não	4	JOIN-2	57,7T
29	Sim	Sim	Não	Não	4	JOIN-1	4T
96	Sim	Sim	Não	Não	7	JOIN-2	123T
48	Sim	Sim	Não	Não	4	JOIN-1	2T
27	Sim	Sim	Não	Não	3	JOIN-2	2T
7	Sim	Sim	Não	Não	4	JOIN-1	4,3T
19	Não	Sim	Não	Não	3	JOIN-1	289G
15	Sim	Sim	Não	Não	6	JOIN-4	10T
17	Sim	Sim	Não	Sim	6	JOIN-1	95G

Conforme a Tabela 5.9, não foi possível notar características predominantes que possam justificar o desempenho com aceleração e desaceleração destas consultas. Isto porque, em ambos os grupos de consultas com aceleração (Grupo 1) e desaceleração (Grupo 2) há Agregação, *Order By*, nenhuma delas contém a cláusula *Distinct*, apenas a Consulta 17 utiliza a operação *Substring*, não há diferença predominante entre quantidade de junções em ambos os grupos, assim como entre as cardinalidades das maiores junções.

Tabela 5.10: Relação de resultados de todas as consultas da carga de trabalho

Consulta	Cenários: # máquinas e tamanho da Base de Dados									
	2 (20GB)		4 (20GB)		4 (40GB)		8 (40GB)		8 (60GB)	
	↑	↓	↑	↓	↑	↓	↑	↓	↑	↓
7	-	14,73%	**28,60%**	-	-	8,49%	-	58,86%	-	17%
15	-	20,85%	**81,10%**	-	-	16,26%	**64,60%**	-	5,51%	18,17%
17	-	36,34%	-	58,11%	7,10%	25,17%	-	36,64%	-	30%
19	-	13,35%	-	15,92%	-	13,26%	-	19,42%	-	18,03%
25	**34,55%**	-	4,78%	5,59%	**54,92%**	7,43%	**24,87%**	3,62%	**51,96%**	-
26	-	16,65%	-	16,59%	2,40%	6,68%	-	81,90%	**31,28%**	-
27	-	8,72%	-	8,90%	-	6,23%	-	10,55%	-	7,58%
29	-	14,66%	**36,07%**	-	-	13,74%	-	30,48%	**31,27%**	-
48	**63,13%**	-	-	64,27%	**25,49%**	1,18%	-	22,44%	**26,74%**	4,27%
91	-	15,43%	**27,25%**	-	0,10%	11,76%	**20,28%**	4,20%	**52,88%**	-
96	**50,09%**	-	**33,07%**	-	**41,96%**	-	**20,39%**	-	**37,31%**	-

A Tabela 5.10 apresenta os resultados mais significantes de cada consulta nos cenários de quantidades de máquinas e bases de dados utilizadas neste experimento. Para cada coluna de cenários de quantidade de máquina e base de dados, há duas colunas que apresentam os resultados mais significantes de aceleração (↑) e desaceleração (↓) comparando cenários *Avaliação* com cenários *Baseline*. As consultas destacadas em cinza apresentaram aceleração em ao menos um cenário *Avaliação* e representam o Grupo 1 com 8 consultas (Consultas 7, 15,

25, 26, 29, 48, 91 e 96). Para estas consultas, os resultados significativos com aceleração estão destacados em negrito. As demais consultas apresentaram resultados mais significantes com desaceleração e representam o Grupo 2 (Consultas 17, 19 e 27) com 3 consultas.

Algumas consultas, como mostra a Tabela 5.10, apresentaram resultados de aceleração e desaceleração para alguma quantidade de máquinas, tais como Consulta 15 (8 (60GB)), Consulta 17 (4 (40GB)), Consulta 25 (4 (20GB), 4 (40GB) e 8 (40GB)), Consulta 26 (4 (40GB)), Consulta 48 (4 (40GB) e 8 (60GB)) e Consulta 91 (4 (40GB) e 8 (40GB)). Neste caso, consideramos o maior resultado entre aceleração e desaceleração para classificar a consulta como Grupo 1 (consultas com aceleração) ou Grupo 2 (consultas com desaceleração).

Com base na Tabela 5.10, é possível notar que as Consultas 19 e 27 não apresentaram nenhum cenário com aceleração. Da mesma forma, a Consulta 17 apresentou apenas um cenário *Avaliação* com desprezível aceleração de 7,1%. Por outro lado, entre as consultas que apresentaram aceleração, as Consultas 7 e 26 apresentaram apenas um cenário com aceleração, sendo 28,6% (4 (20GB)) para a primeira e 31,28% (8 (60GB)) para a segunda. A Consulta 48 não apresentou maior aceleração que desaceleração para os experimentos com cenários de 4 máquinas (20GB) e cenários de 8 máquinas (40GB). As Consultas 25 e 91 não apresentaram maior aceleração que desaceleração em apenas um experimento, sendo cenários de 4 máquinas (20GB) para a primeira e cenários de 4 máquinas (40GB) para a segunda. A Consulta 96, por sua vez, apresentou, exclusivamente, aceleração em todos os casos.

5.6.3 CARGA DE TRABALHO

Esta seção apresenta os resultados e avaliação destes para as cargas de trabalho de cada quantidade de máquinas. O resultado de cada carga de trabalho é formado pela soma dos resultados de todas as consultas em cada uma das 5 rodadas, eliminando as rodadas com menor e maior valor, e realizando a média das 3 rodadas restantes. Tendo em vista que as Consultas 15, 17, 25 e 96 apresentaram problemas na execução sob o cenário com 64 nós totais (*m:8_dn:8_wn:56*), os resultados para a carga de trabalho com 8 máquinas não contêm valor para este cenário. A Figura 5.38 ilustra estes resultados.

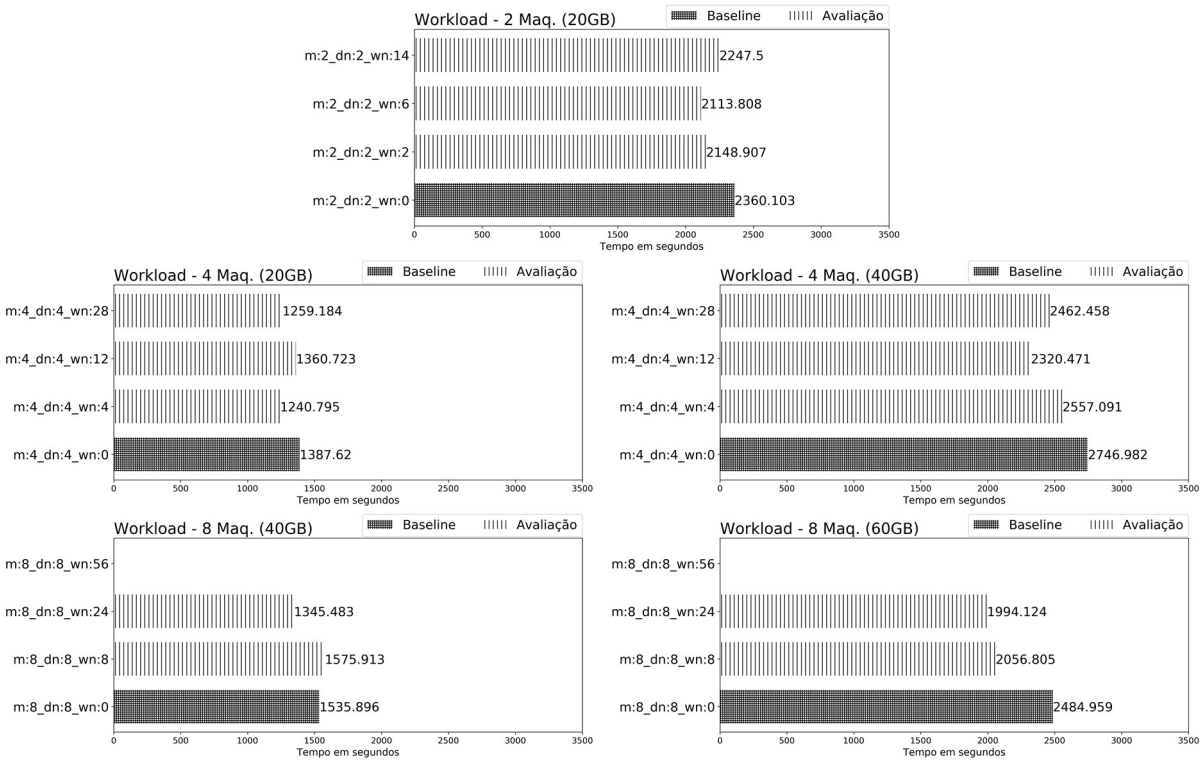

Figura 5.38: Resultados para Cargas de Trabalho

Os resultados para o processamento das cargas de trabalho apresentam uma característica predominante (vista também em algumas consultas): o acréscimo de *worker nodes* causa aceleração e depois desaceleração a partir de certo ponto. Estes resultados mostram que os cenários que utilizam 50% dos recursos disponíveis apresentaram o melhor desempenho e os cenários que utilizam 100% dos recursos disponíveis apresentaram desaceleração em relação ao melhor caso.

Para cenários com 2 máquinas (20GB) o cenário *Avaliação m:2_dn:2_wn:6* apresentou aceleração de 11,65% em relação ao cenário *Baseline m:2_dn:2_wn:0*. Por outro lado, o cenário *Avaliação m:2_dn:2_wn:14* apresentou aceleração inferior de 5,01% em relação ao cenário *Baseline m:2_dn:2_wn:0* e desaceleração de 6,32% em relação ao cenário *Avaliação m:2_dn:2_wn:6.* Para cenários com 4 máquinas (20GB), cenários *Avaliação* apresentaram aceleração de até 11,83% com o cenário *m:4_dn:4_wn:4* e aceleração inferior de 10,2% com o cenário *m:4_dn:4_wn:28* em relação ao cenário *Baseline m:4_dn:4_wn:0.*

Para cenários com 4 máquinas (40GB) o cenário *Avaliação m:4_dn:4_wn:12* apresentou aceleração de 18,31% em relação ao cenário *Baseline m:4_dn:4_wn:0*. Por outro lado, o cenário *Avaliação m:4_dn:4_wn:28* apresentou aceleração inferior de 11,55% em relação ao cenário *Baseline m:4_dn:4_wn:0* e desaceleração de 6,12% em relação ao cenário *Avaliação m:4_dn:4_wn:12.* Para cenários com 8 máquinas o cenário *Avaliação m:8_dn:8_wn:24*

apresentou aceleração de 14,15% (40GB) e 24,61% (60GB) em relação ao cenário *Baseline m:8_dn:8_wn:0.* A Figura 5.39 apresenta os principais resultados de aceleração e desaceleração para a carga de trabalho.

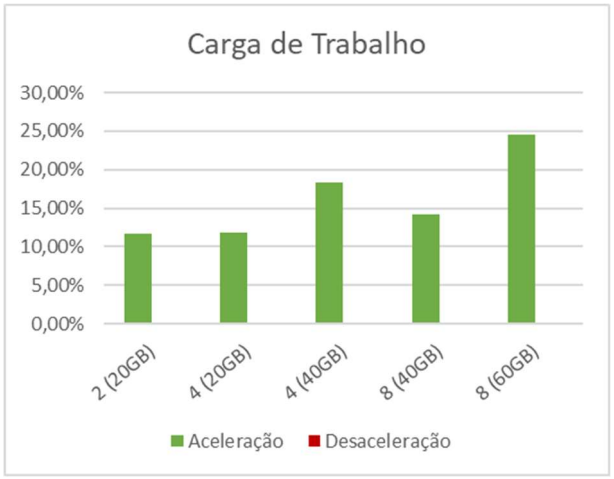

Figura 5.39: Principais resultados para Carga de Trabalho

Tendo em vista que a Figura 5.38 não apresenta resultado para o cenário *Avaliação m:8_dn:8_wn:56* do experimento com 8 máquinas devido aos problemas na execução de algumas consultas, retiramos tais consultas do cálculo do resultado da carga de trabalho para esta quantidade de máquinas. Desta forma, a Figura 5.40 apresenta resultados do processamento da carga de trabalho sob cenários com 8 máquinas desconsiderando os resultados das Consultas 15, 17, 25 e 96.

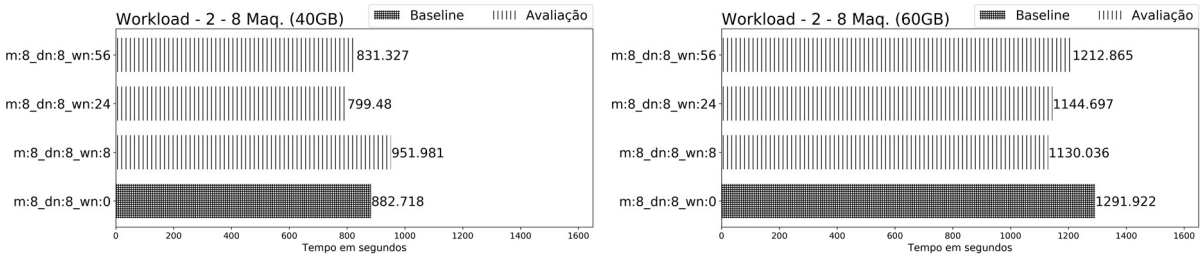

Figura 5.40: Resultados para a Carga de Trabalho reduzida com 8 máquinas

Com a retirada das consultas que apresentaram problemas durante a execução, os gráficos resultantes do desempenho dos cenários com 8 máquinas nesta carga de trabalho apresentaram a mesma característica predominante para 2 e 4 máquinas: o acréscimo de *worker nodes* causa aceleração e desaceleração a partir de certo ponto. Logo, considerando a base de dados 60GB, o cenário *Avaliação* de melhor desempenho (*m:8_dn:8_wn:8*) apresentou aceleração de 14,33% em relação ao cenário *Baseline m:8_dn:8_wn:0*. Por outro lado, o cenário *Avaliação m:8_dn:8_wn:54* apresentou aceleração inferior de 6,52% em relação ao cenário

Baseline m:8_dn:8_wn:0 e desaceleração de 7,33% em relação ao cenário *Avaliação m:8_dn:8_wn:8.*

5.7 CONSIDERAÇÕES FINAIS

A partir da avaliação dos resultados dos experimentos realizados neste capítulo, foi possível notar que os resultados mais significantes para o impacto do uso de *data* e *worker nodes* no processamento de cada consulta gerou dois grupos: Grupo 1 como sendo consultas que apresentaram aceleração (Consultas 7, 15, 25, 26, 29, 48, 91 e 96) e Grupo 2 como sendo consultas que apresentaram desaceleração (Consultas 17, 19 e 27). O melhor desempenho para o Grupo 1 foi a Consulta 96 que apresentou exclusivamente aceleração em todos os casos. Considerando cenários com 8 máquinas (60GB), o melhor caso foi a Consulta 91 com aceleração de até 52,88%. Em relação ao Grupo 2, o pior resultado foi desaceleração de 30% (Consulta 17). Identificamos também que as consultas 15, 17, 25 e 96 não apresentaram resultados para o cenário *m:8_dn:8_wn:56* (64 nós totais) devido a consecutivos erros de estouro de memória. Estes erros são justificados pela limitação de memória principal (16GB) de cada máquina do *cluster* e pelo grande volume de dados utilizado, uma vez que cada nó utilizou no máximo 2GB de memória principal durante a execução das consultas.

Com relação ao Grupo 1, identificamos que para o processamento das consultas em cenários de ao menos uma quantidade de máquina (2, 4 ou 8) apresentou um comportamento predominante: o acréscimo de *worker nodes* causa aceleração seguida de desaceleração no tempo de processamento. Este comportamento também foi identificado em todos os casos para os resultados das cargas de trabalho, assim como para a Consulta C2 do Experimento I. Isto mostra que há uma curva que oscila entre aceleração e desaceleração com relação à adição de *worker nodes*. Com relação ao Grupo 2, não identificamos melhora no desempenho do processamento das consultas, que apresentaram desaceleração gradativa conforme a adição de *worker nodes* para todas as consultas.

Estes resultados se apresentam promissores pois mostram que a adição indefinida de *worker nodes* não é uma garantia de melhora no desempenho do processamento da consulta e/ou carga de trabalho. Esta adição pode causar aceleração seguida de desaceleração (Grupo 1 e cargas de trabalho) ou mesmo gradativa desaceleração (Grupo 2). Neste contexto, as consultas integrantes da carga de trabalho utilizada neste experimento não nos permitiram chegar a conclusões a respeito de um limite para a adição de *worker nodes* que venha a proporcionar melhora no tempo de processamento.

O próximo capítulo (Capítulo 6) apresenta as Considerações Finais desta dissertação.

CAPÍTULO 6 – CONCLUSÃO

Esta dissertação avaliou o impacto do uso de *data nodes*, nós que processam e armazenam dados, e *worker nodes*, nós que realizam processamento e não armazenam dados, no processamento paralelo de consultas SQL em *cluster*. Em nossa revisão bibliográfica, a maioria dos sistemas de gerência de Big Data avaliados gerencia cada máquina como sendo um tipo de nó. Tais sistemas exploram a tecnologia *multi-core* através do processamento *multi-thread* ou atribuindo vários nós para uma mesma máquina, onde o recurso de armazenamento é fatiado ou compartilhado. Contudo, dentre todas as abordagens apresentadas, nenhuma explora totalmente os recursos disponíveis de armazenamento (discos de dados) e processamento (núcleos de CPU) de forma independente sem acesso concorrente, onde várias atividades podem ser gerenciadas e processadas em paralelo, por nós independentes (*data nodes* e *worker nodes*), em uma mesma máquina através do paralelismo intra-máquina.

Para avaliar o comportamento da abordagem do uso de *data nodes* e *worker nodes* alocados em uma mesma máquina, planejamos, executamos e avaliamos dois experimentos (Experimento I e II) utilizando como plataforma base o mecanismo de execução paralela de consulta relacional MyriaX. A escolha do MyriaX foi motivada por este mecanismo permitir o uso de *data nodes* e *worker nodes* de forma independente em uma mesma máquina, apesar de isto nunca ter sido explorado. Todos os *scripts* e base de dados que utilizamos em ambos os experimentos estão disponíveis na plataforma GitHub[20].

Em ambos os experimentos, realizamos o processamento de consultas com características diferentes sob diversas configurações de alocações de *data nodes* e *worker nodes*. Nomeamos estas configurações como cenários. Nomeamos também os cenários padrões utilizados pelo MyriaX como sendo *Baseline* e cenários da nossa abordagem como sendo *Avaliação*. Além disso, utilizamos um *script* para otimizar as implantações em cada configuração e capturar o tempo de processamento, além de calcular a média e salvar os resultados em arquivo no formato JSON. O *cluster* utilizado para os experimentos, nomeado Oscar, é de propriedade do Instituto de Computação da Universidade Federal Fluminense.

O Experimento I teve como finalidade confirmar a hipótese de que é possível diminuir o tempo de processamento de consultas através da adição de *worker nodes* em núcleos ociosos de processadores. Para tal, experimentos foram realizados com dois tipos de consultas: uma de auto-junção (C1) e outra para identificação de triângulos (C2), utilizando uma base de dados

[20] https://dew-uff.github.io/myria-uff/

do Twitter, sob diversos cenários em *cluster*. Os resultados mostram que, para ambas consultas, sempre há pelo menos um cenário *Avaliação* com melhor desempenho que o cenário *Baseline*. Assim, conseguimos confirmar a hipótese deste experimento. No melhor caso, ao processar a consulta C1, explorando todos os recursos de núcleos de processamento por meio de *worker nodes*, conseguimos aceleração de até 2,92x. Entretanto, nem sempre a adição gradativa de *worker nodes* apresentou melhores resultados; em quase todos os casos há uma deterioração a partir de um determinado ponto, para a consulta C2, devido às suas diferentes características.

O Experimento II avaliou o desempenho da abordagem do uso de *data nodes* e *worker nodes* no processamento de cargas de trabalhos reais que incluem um conjunto de consultas com diversas características. Para tal, após avaliar um conjunto de benchmarks, selecionamos o TPC-DS, pois trata-se de uma evolução do TPC-H para Big Data com maior complexidade e utiliza apenas dados estruturados. A carga de trabalho deste experimento é composta por 11 consultas selecionadas do conjunto de 99 disponíveis para este benchmark. Para tornar esta carga de trabalho compatível com o MyriaX, utilizamos a ferramenta de gerência de banco de dados SQL Server Management Studio para gerar planos otimizados que foram convertidos para o formato JSON.

Devido às limitações de infraestrutura do *cluster* utilizado, no Experimento II os cenários de *data nodes* e *worker nodes* foram divididos por quantidade de máquinas (2, 4 e 8) e processados sobre bases de dados do TPC-DS com escalas de geração diferentes (20GB, 40GB e 60GB). Sendo assim, avaliamos os resultados individuais de cada consulta e também da carga de trabalho. A partir desta avaliação, foi possível identificar dois grupos de consultas: Grupo 1 (8 consultas) como sendo consultas que apresentaram aceleração e Grupo 2 (3 consultas) como sendo consultas que apresentaram desaceleração.

O melhores desempenhos para o Grupo 1 foram: aceleração de até 63,13% em cenários com 2 máquinas (20GB) com a Consulta 48, aceleração de até 81,1% em cenários com 4 máquinas (20GB) com a Consulta 15, aceleração de até 54,92% em cenários com 4 máquinas (40GB) com a Consulta 25, aceleração de até 64,6% em cenários com 8 máquinas (40GB) com a Consulta 15 e aceleração de até 52,88% em cenários com 8 máquinas (60GB) com a Consulta 91. Para o Grupo 2, os piores resultados foram: desaceleração de 58,11% em cenários com 4 máquinas (20GB) com a Consulta 17 e desaceleração de 19,42% em cenários com 8 máquinas (40GB) com a Consulta 19.

Identificamos que, para o processamento das consultas em cenários de ao menos uma quantidade de máquina (2, 4 ou 8) do Grupo 1, para cargas de trabalho e também para a Consulta C2 do Experimento I, há um comportamento predominante: o acréscimo de *worker nodes* causa

aceleração seguida de desaceleração no tempo de processamento. Isto mostra que há uma curva que oscila entre aceleração e desaceleração com relação à adição de *worker nodes*.

Os resultados para ambos os experimentos apresentam melhora no desempenho seguida de uma deterioração a partir de um determinado ponto com o acréscimo de *worker nodes*. Tais resultados podem ser considerados importantes, pois mostram que adicionar nós indefinidamente não é garantia de melhora no tempo de processamento. Descobrir qual é este limite é um trabalho promissor e abre oportunidades para elaboração de heurísticas que possam ser usadas na geração do plano da consulta e adição de *worker nodes*.

Durante o período experimental desta dissertação, o *cluster* utilizado recebeu manutenção e peças foram migradas de nós inutilizados para nós ativos. Além disso, processos de manutenção do sistema operacional, os quais não controlamos, podem ter sido acionados durante a execução dos experimentos. Não há como saber se estes casos podem ter influenciado o tempo de processamento das consultas de ambos os experimentos. Buscamos mitigar estas ameaças à validade de nossos experimentos por meio da seleção de nós homogêneos e múltiplas execuções das consultas. Sendo assim, seria necessário executar novamente os experimentos em outro ambiente para obtermos conclusões concretas sobre esta possível influência.

O conjunto de consultas utilizado nos Experimento I e II não nos permitiu chegar a conclusões sobre o limite do uso de *worker nodes*. Sendo assim, como trabalhos futuros pretendemos analisar um conjunto maior de consultas buscando confirmar quais são as características das consultas que apresentam aceleração ou desaceleração. Variáveis como tráfego de rede, taxa de I/O, memória disponível e quantidade de processos em execução também serão avaliadas durante a execução dos experimentos. Além disso, avaliaremos a influência de memória *cache* no Experimento II e o uso de outras estratégias de particionamento.

Neste contexto, pretende-se tentar chegar a conclusões que nos auxiliem a identificar o ponto ótimo, ou o mais próximo dele, entre a característica da consulta e a quantidade de *worker nodes*, a quantidade de máquinas e o volume de dados no processamento paralelo de consultas relacionais. A melhor relação entre estas variáveis permite a elaboração de heurísticas sobre elasticidade de recurso, em que um otimizador pode alocar o melhor cenário de *worker nodes* e *data nodes* dada a característica de uma consulta. Estas conclusões serão utilizadas na elaboração de heurísticas a serem implementadas em mecanismos de otimização de recursos para sistemas de gerência de Big Data.

REFERÊNCIAS

ABOUZEID, Azza et al. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *Proceedings of the VLDB Endowment (PVLDB)*, v. 2, n. 1, p. 922–933, 2009.

ALEXANDROV, Alexander et al. Massively Parallel Data Analysis With PACTs on Nephele. *VLDB Endowment*, v. 3, n. 1–2, p. 1625–1628, 2010.

ALSUBAIEE, Sattam et al. ASTERIX: Scalable Warehouse-Style Web Data Integration. *International Workshop on Information Integration on the Web*, p. 1–4, 2012.

ALSUBAIEE, Sattam et al. AsterixDB: A scalable, open source BDMS. *Proceedings of the VLDB Endowment*, v. 7, n. 14, p. 1905–1916, 2014.

ANANTHANARAYANAN, Rajagopal et al. Photon: Fault-tolerant and Scalable Joining of Continuous Data Streams. 2013, [S.l: s.n.], 2013. p. 577–588.

ARMBRUST, Michael et al. Spark SQL: Relational Data Processing in Spark. 2015, [S.l.]: ACM Press, 2015. p. 1383–1394. Disponível em: <http://dl.acm.org/citation.cfm?doid=2723372.2742797>. Acesso em: 28 jul 2015.

BAPPALIGE, Sachin P. *An Introduction to Apache Hadoop for Big Data*. Disponível em: <https://opensource.com/life/14/8/intro-apache-hadoop-big-data>.

BITTORF, MKABV et al. Impala: A Modern, Open-Source SQL Engine for Hadoop. 2015, [S.l: s.n.], 2015.

BORKAR, Vinayak et al. Hyracks: A Flexible and Extensible Foundation for Data-intensive Computing. 2011, [S.l: s.n.], 2011. p. 1151–1162.

CARBONE, Paris et al. Apache Flink™: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin Issues*, IEEE Data Engineering Bulletin Issues, 2015. , p. 28–38.

COOPER, Brian F. et al. Benchmarking Cloud Serving Systems with YCSB. 2010, New York, NY, USA. Anais... New York, NY, USA: ACM, 2010. p. 143–154.

DAGEVILLE, Benoit et al. The Snowflake Elastic Data Warehouse. *International Conference on Management of Data (SIGMOD)*, p. 215–226, 2016.

DAS, Sudipto; AGRAWAL, Divyakant; EL ABBADI, Amr. ElasTraS: An Elastic, Scalable, and Self-Managing Transactional Database for the Cloud. *Transactions on Database Systems (TODS)*, v. 38, n. 1, p. 5, 2013.

DEAN, Jeffrey; GHEMAWAT, Sanjay. MapReduce: Simplified Data Processing on Large Clusters. *Operating Systems Design and Implementation (OSDI)*, 00000, p. 137–150, 2004.

DEWITT, David; GRAY, Jim. Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, v. 35, n. 6, p. 85–98, 1992.

GARCIA-MOLINA, Hector; ULLMAN, Jeffrey D.; WIDOM, Jennifer. *Database Systems: The Complete Book*. 2 edition ed. Upper Saddle River, N.J: Pearson, 2008. p. 734–735.

GHAZAL, Ahmad et al. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. 2013, [S.l.]: ACM, 2013. p. 1197–1208.

GHEMAWAT, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The Google File System. 2003, [S.l: s.n.], 2003. p. 29–43.

GRAEFE, Goetz. *Encapsulation of Parallelism in the Volcano Query Processing System*. [S.l.]: ACM, 1990. v. 19. (, 2).

GUPTA, Anurag et al. Amazon Redshift and the Case for Simpler Data Warehouses. 2015, [S.l: s.n.], 2015. p. 1917–1923.

HALPERIN, Daniel et al. Demonstration of the Myria Big Data Management Service. *International Conference on Management of Data (SIGMOD)*, p. 881–884, 2014.

HARDAVELLAS, Nikos; PANDIS, Ippokratis. Inter-Query Parallelism. *Encyclopedia of Database Systems*. [S.l.]: Springer, 2009a. p. 1566–1567.

HARDAVELLAS, Nikos; PANDIS, Ippokratis. Intra-Query Parallelism. *Encyclopedia of Database Systems*. [S.l.]: Springer, 2009b. p. 1567–1568.

HO, Ricky. *Pragmatic Programming Techniques*. Disponível em: <http://horicky.blogspot.com/2010/07/google-pregel-graph-processing.html>.

HU, Xiaocheng; TAO, Yufei; CHUNG, Chin-Wan. Massive Graph Triangulation. 2013, [S.l: s.n.], 2013. p. 325–336.

HUANG, Shengsheng et al. The HiBench Benchmark Suite: Characterization of the MapReduce-based Data Analysis. In: DATA ENGINEERING WORKSHOPS (ICDEW), 2010, [S.l.]: IEEE, 2010. p. 41–51.

ISARD, Michael et al. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. 2007, [S.l: s.n.], 2007. p. 59–72.

KIM, Changkyu et al. Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-core CPUs. *Proceedings of the VLDB Endowment (PVLDB)*, v. 2, n. 2, p. 1378–1389, 2009.

KWON, YongChul; BALAZINSKA, Magdalena; GREENBERG, Albert. Fault-tolerant Stream Processing Using a Distributed, Replicated File System. *VLDB Endowment*, v. 1, n. 1, p. 574–585, 2008.

LAMB, Andrew et al. The Vertica Analytic Database: C-store 7 Years Later. *Proceedings of the VLDB Endowment*, v. 5, n. 12, p. 1790–1801, 2012.

LEVENE, Mark; LOIZOU, George. Why is the Snowflake Schema a Good Data Warehouse Design? *Information Systems*, v. 28, n. 3, p. 225–240, 2003.

MALEWICZ, Grzegorz et al. Pregel: A System for Large-scale Graph Processing. 2010, [S.l: s.n.], 2010. p. 135–146.

MATTOSO, Marta et al. ParGRES: Middleware para Processamento Paralelo de Consultas OLAP em Clusters de Banco de Dados. 2005, Uberlandia, Brazil. Anais... Uberlandia, Brazil: [s.n.], 2005. p. 19–24.

MEHTA, Manish; DEWITT, David J. Data Placement in Shared-nothing Parallel Database Systems. *The VLDB Journal*, v. 6, n. 1, p. 53–72, 1997.

MISHRA, Priti; EICH, Margaret H. Join Processing in Relational Databases. *ACM Computing Surveys (CSUR)*, v. 24, n. 1, p. 63–113, 1992.

NAMBIAR, Raghunath Othayoth; POESS, Meikel. The Making of TPC-DS. 2006, [S.l.]: VLDB Endowment, 2006. p. 1049–1058.

NEUMEYER, Leonardo et al. S4: Distributed Stream Computing Platform. 2010, [S.l: s.n.], 2010. p. 170–177.

PIVOTAL SOFTWARE. *Introduction to the Greenplum Database Architecture*. Disponível em: <https://greenplum.org/gpdb-sandbox-tutorials/introduction-greenplum-database-architecture/#ffs-tabbed-12>. Acesso em: 11 out 2018.

POESS, Meikel et al. TPC-DS, Taking Decision Support Benchmarking to the Next Level. 2002, [S.l: s.n.], 2002. p. 582–587.

POESS, Meikel; NAMBIAR, Raghunath Othayoth; WALRATH, David. Why You Should Run TPC-DS: A Workload Analysis. 2007, [S.l: s.n.], 2007. p. 1138–1149.

SCHNEIDER, Donovan A.; DEWITT, David J. A performance Evaluation of Four Parallel Join Algorithms in a Shared-nothing Multiprocessor Environment. *International Conference on Management of Data (SIGMOD)*, v. 18, p. 110–121, 1989.

SILVA, Frank Willian Rodrigues; ALMEIDA, Victor Teixeira; BRAGANHOLO, Vanessa. Explorando Arquiteturas Multi-core para Processamento Eficiente de Consultas em Sistemas de Gerência de Big Data. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS (SBBD), 2017, Uberlândia, Brazil. Anais... Uberlândia, Brazil: [s.n.], 2017.

SOLIMAN, Mohamed A. et al. Orca: A Modular Query Optimizer Architecture for Big Data. 2014, [S.l.]: ACM, 2014. p. 337–348.

STONEBRAKER, Michael. The Case for Shared Nothing. *IEEE Database Engineering*, v. 9, n. 1, p. 4–9, 1986.

THUSOO, Ashish et al. Hive: A Warehousing Solution Over a Map-Reduce Framework. *VLDB Endowment*, v. 2, n. 2, p. 1626–1629, 2009.

TOSHNIWAL, Ankit et al. Storm@Twitter. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA (SIGMOD), 2014, [S.l: s.n.], 2014. p. 147–156.

VERTICA. *Vertica*. Disponível em: <https://www.vertica.com/>. Acesso em: 10 nov 2018.

WANG, Jingjing et al. The Myria Big Data Management and Analytics System and Cloud Service. 2017, [S.l: s.n.], 2017. p. 37–47.

WANG, Lei et al. Bigdatabench: A Big Data Benchmark Suite From Internet Services. 2014, [S.l.]: IEEE, 2014. p. 488–499.

WARNEKE, Daniel; KAO, Odej. Nephele: Efficient Parallel Data Processing in the Cloud. 2009, [S.l: s.n.], 2009. p. 1–10.

WU, Yingjun; TAN, Kian-Lee. ChronoStream: Elastic Stateful Stream Computation in the Cloud. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING (ICDE), 2015, [S.l: s.n.], 2015. p. 723–734.

ZAHARIA, Matei et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. *Networked Systems Design and Implementation (NSDI)*, p. 15–28, 2012.

ZHANG, Jianyong et al. Synthesizing Representative I/O Workloads for TPC-H. 2004, [S.l.]: IEEE, 2004. p. 142–142.

ANEXO A – RELAÇÃO ENTRE CONSULTA E MOTIVO DE DESCARTE DAS CONSULTAS DO TPC-DS

A carga de trabalho utilizada nos experimentos do Capítulo 5 contém 11 consultas selecionadas do conjunto de 99 disponíveis no benchmark TPC-DS. Esta seleção considerou a complexidade das consultas e a limitação dos operadores reconhecidos pelo mecanismo de execução de consulta MyriaX. Assim, selecionamos consultas com alta complexidade, que abrangem tabelas fato e dimensões, que utilizam operadores relacionais reconhecidos pelo MyriaX. Os motivos de descarte das demais consultas são apresentados nas tabelas abaixo, onde as consultas selecionadas estão destacadas em cinza. A Tabela A.1 relaciona as consultas de 1 a 26, a Tabela A.2 as consultas entre 27 e 71 e a Tabela A.3 as consultas entre 72 e 99.

Tabela A.1: Seleção ou motivos de descarte das consultas de 1 a 26

Consulta	Consulta Selecionada ou Motivo de descarte
1	Contém operador WITH e sub consultas.
2	Contém operadores WITH, CASE e sub consultas.
3	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 3 tabelas, sendo uma tabela fato.
4	Contém operadores WITH, CASE e sub consultas.
5	Contém operador WITH e sub consultas.
6	Contém operador HAVING e sub consultas.
7	Consulta selecionada.
8	Contém operador HAVING e sub consultas.
9	Contém operador CASE e sub consultas.
10	Contém operador EXISTS e sub consultas.
11	Contém operadores WITH, CASE e sub consultas.
12	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 3 tabelas, sendo uma tabela fato.
13	Consulta selecionada na primeira seleção, porém descartada pela complexidade na conversão para o formato JSON.
14	Contém operadores WITH, INTERSECT, HAVING e sub consultas.
15	Consulta selecionada.
16	Contém operadores EXISTS, NOT EXISTS e sub consultas.
17	Consulta selecionada.
18	Consulta selecionada na primeira seleção, porém descartada por apresentar problemas na execução durante os testes iniciais.
19	Consulta selecionada.
20	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 3 tabelas, sendo uma tabela fato.
21	Contém operador CASE e sub consultas.
22	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 4 tabelas, sendo uma tabela fato.
23	Contém operadores WITH, HAVING e sub consultas.
24	Contém operadores WITH, HAVING e sub consultas.
25	Consulta selecionada.
26	Consulta selecionada.

Tabela A.2: Seleção ou motivos de descarte das consultas de 27 a 71

Consulta	Consulta Selecionada ou Motivo de descarte
27	Consulta selecionada.
28	Contém sub consultas.
29	Consulta selecionada.
30	Contém operador WITH e sub consultas.
31	Contém operadores WITH, CASE e sub consultas.
32	Contém sub consulta.
33	Contém operador WITH e sub consultas.
34	Contém operador CASE e sub consultas.
35	Contém operador EXISTS e sub consultas.
36	Contém operadores OVER e CASE e função RANK.
37	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 4 tabelas, sendo uma tabela fato.
38	Contém operador INTERSECT e sub consultas.
39	Contém operadores WITH, CASE e sub consultas.
40	Contém operador CASE.
41	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 1 tabela.
42	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 3 tabelas, sendo uma tabela fato.
43	Contém operador CASE.
44	Contém operadores OVER, HAVING, função RANK e sub consultas.
45	Contém sub consulta.
46	Contém sub consulta.
47	Contém operadores WITH, OVER e CASE, função RANK e sub consultas.
48	Consulta selecionada.
49	Contém operador OVER, função RANK e sub consultas.
50	Contém operador CASE.
51	Contém operadores WITH, CASE e sub consultas.
52	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 3 tabelas, sendo uma tabela fato.
53	Contém operadores OVER, CASE e sub consultas.
54	Contém operador WITH e sub consultas.
55	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 3 tabelas, sendo uma tabela fato.
56	Contém operador WITH e sub consultas.
57	Contém operadores WITH, OVER e CASE, função RANK e sub consultas.
58	Contém operador WITH e sub consultas.
59	Contém operadores WITH, CASE e sub consultas.
60	Contém operador WITH e sub consultas.
61	Contém sub consulta.
62	Contém operador CASE.
63	Contém operadores OVER, CASE e sub consultas.
64	Contém operadores WITH, HAVING e sub consultas.
65	Contém sub consulta.
66	Contém operador CASE e sub consultas.
67	Contém operador OVER, função RANK e sub consultas.
68	Contém sub consulta.
69	Contém operadores EXISTS, NOT EXISTS e sub consultas.
70	Contém operadores OVER e CASE, função RANK e sub consultas.
71	Contém sub consulta.

Tabela A.3: Seleção ou motivos de descarte das consultas de 72 a 99

Consulta	Consulta Selecionada ou Motivo de descarte
72	Contém operador CASE.
73	Contém operador CASE e sub consulta.
74	Contém operadores WITH, CASE e sub consulta.
75	Contém operadores WITH, COALESCE e sub consulta.
76	Contém sub consultas.
77	Contém operador WITH e sub consultas.
78	Contém operadores WITH, COALESCE e sub consultas.
79	Contém sub consulta.
80	Contém operadores WITH, COALESCE e sub consultas.
81	Contém operador WITH e sub consultas.
82	Consulta selecionada na primeira seleção, porém descartada por relacionar apenas 4 tabelas, sendo uma tabela fato.
83	Contém operador WITH e sub consultas.
84	Consulta selecionada na primeira seleção, porém descartada por conter concatenação de caracteres na seleção.
85	Consulta selecionada na primeira seleção, porém descartada pela complexidade na conversão para o formato JSON.
86	Contém operadores OVER e CASE e função RANK.
87	Contém sub consultas.
88	Contém sub consultas.
89	Contém operadores OVER, CASE e sub consultas.
90	Contém sub consulta.
91	Consulta selecionada.
92	Contém operador INTERVAL e sub consulta.
93	Contém operador CASE e sub consulta.
94	Contém operadores EXISTS, NOT EXISTS, INTERVAL e sub consultas.
95	Contém operadores WITH, INTERVAL e sub consultas.
96	Consulta selecionada.
97	Contém operadores WITH, CASE e sub consulta.
98	Contém operadores OVER, PARTITION BY, INTERVAL e sub consultas.
99	Contém operador CASE.

ANEXO B – VALORES DE CARDINALIDADES DOS PLANOS OTIMIZADOS DO EXPERIMENTO II

Os planos otimizados das consultas integrantes da carga de trabalho utilizada no Experimento II, gerados pela ferramenta SQL Server Management Studio e apresentados no Capítulo 5, contêm valores de cardinalidades referentes à Base 3 (fator de geração 60GB) do TPC-DS. Calculamos e tabulamos as cardinalidades e resultados de cada passo destes planos para Base 1 (fator de geração 20GB) e Base 2 (fator de geração 40GB) também. Nesta seção, são apresentadas tabelas com os valores para cardinalidades e resultados referentes a Base 1, Base 2 e Base 3. Cada linha destas tabelas representa um passo do referente plano otimizado.

Tabela B.1: Cardinalidades e resultados para o plano otimizado da Consulta 7

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ customer_demographics	Filtro de predicados (PF-1)	-	-	-	27.440	27.440	27.440
PF-1 ⋈ store_sales	Junção (JOIN-1)	1.580.514.694.080	3.161.181.844.800	4.741.651.509.840	784.871	1.567.738	2.352.367
σ date_dim	Filtro de predicados (PF-2)	-	-	-	365	365	365
JOIN-1 ⋈ PF-2	Junção (JOIN-2)	286.477.915	572.224.370	858.613.955	151.696	303.547	458.605
σ store	Filtro de predicados (PF-3)	-	-	-	44	34	50
JOIN-2 ⋈ PF-3	Junção (JOIN-3)	6.674.624	10.320.598	22.930.250	149.864	105.095	126.707
JOIN-3 ⋈ item	Junção (JOIN-4)	4.196.192.000	5.464.940.000	9.376.318.000	149.864	105.095	126.707
# Tuplas Resultantes					14.001	25.557	35.795

Tabela B.2: Cardinalidades e resultados para o plano otimizado da Consulta 15

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ customer_address	Filtro de predicados (PF-1)	-	-	-	32.870	74.359	115.836
σ customer	Filtro de predicados (PF-2)	-	-	-	129.364	290.571	451.509
PF-1 ⋈ PF-2	Junção (JOIN-1)	4.252.194.680	21.606.568.989	52.300.649.016	32.060	71.997	112.360
JOIN-1 ⋈ cd2	Junção (JOIN-2)	61.580.848.000	138.291.837.600	215.821.088.000	31.477	70.711	110.308
catalog_sales ⋈ item	Junção (JOIN-3)	806.500.324.000	2.994.992.468.000	6.393.731.276.000	28.803.583	57.596.009	86.401.774
σ cd1	Filtro de predicados (PF-3)	-	-	-	137.200	137.200	137.200
JOIN-3 ⋈ PF-3	Junção (JOIN-4)	3.951.851.587.600	7.902.172.434.800	11.854.323.392.800	2.046.585	4.084.323	6.137.449
JOIN-2 ⋈ JOIN-4	Junção (JOIN-5)	64.420.356.045	288.806.563.653	677.009.724.292	242.079	478.843	724.535
σ date_dim	Filtro de predicados (PF-4)	-	-	-	365	365	365
JOIN-5 ⋈ PF-4	Junção (JOIN-6)	88.358.835	174.777.695	264.455.275	47.403	95.442	144.875
# Tuplas Resultantes					47.301	95.249	144.504

Tabela B.3: Cardinalidades e resultados para o plano otimizado da Consulta 17

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ customer_demographics	Filtro de predicados (PF-1)	-	-	-	109.760	109.760	109.760
customer ⋈ PF-1	Junção (JOIN-1)	29.196.160.000	65.856.000.000	102.406.080.000	14.825	33.338	51.585
σ household_demographics	Filtro de predicados (PF-2)	-	-	-	1.200	1.200	1.200
JOIN-1 ⋈ PF-2	Junção (JOIN-2)	17.790.000	40.005.600	61.902.000	2.416	5.429	8.369
JOIN-2 ⋈ catalog_returns	Junção (JOIN-3)	6.963.233.328	31.269.112.705	72.308.745.830	25.513	50.648	75.678
σ customer_address	Filtro de predicados (PF-3)	-	-	-	58.641	132.335	205.601
JOIN-3 ⋈ PF-3	Junção (JOIN-4)	1.496.107.833	6.702.503.080	15.559.472.478	11.078	22.551	32.660
σ date_dim	Filtro de predicados (PF-4)	-	-	-	31	31	31
JOIN-4 ⋈ PF-4	Junção (JOIN-5)	343.418	699.081	1.012.460	190	363	530
JOIN-5 ⋈ call_center	Junção (JOIN-6)	1.140	2.904	4.240	188	360	524
# Tuplas Resultantes					6	10	10

Tabela B.4: Cardinalidades e resultados para o plano otimizado da Consulta 19

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ time_dim	Filtro de predicados (PF-1)	-	-	-	1.800	1.800	1.800
PF-1 ⋈ store_sales	Junção (JOIN-1)	103.678.077.600	207.366.156.000	311.041.279.800	1.091.595	2.194.731	3.289.286
σ household_demographics	Filtro de predicados (PF-2)	-	-	-	720	720	720
JOIN-1 ⋈ PF-2	Junção (JOIN-2)	785.948.400	1.580.206.320	2.368.285.920	108.080	216.700	325.725
σ store	Filtro de predicados (PF-3)	-	-	-	6	15	24
JOIN-2 ⋈ PF-3	Junção (JOIN-3)	648.480	3.250.500	7.817.400	18.582	29.918	43.268
# Tuplas Resultantes					1	1	1

Tabela B.5: Cardinalidades e resultados para o plano otimizado da Consulta 25

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ d1	Filtro de predicados (PF-1)	-	-	-	30	30	30
store_sales ⋈ PF-1	Junção (JOIN-1)	1.727.967.960	3.456.102.600	5.184.021.330	521.217	1.040.537	1.562.234
JOIN-1 ⋈ store_returns	Junção (JOIN-2)	2.999.653.350.615	11.981.414.164.365	26.982.018.299.088	39.037	77.370	116.314
σ d2	Filtro de predicados (PF-2)	-	-	-	122	122	122
JOIN-2 ⋈ PF-2	Junção (JOIN-3)	4.762.514	9.439.140	14.190.308	22.575	44.616	66.982
JOIN-3 ⋈ catalog_sales	Junção (JOIN-4)	650.240.886.225	2.569.703.537.544	5.787.363.626.068	147	284	319
σ d3	Filtro de predicados (PF-3)	-	-	-	1.096	1.096	1.096
JOIN-4 ⋈ PF-3	Junção (JOIN-5)	161.112	311.264	349.624	83	104	121
JOIN-5 ⋈ item	Junção (JOIN-6)	2.324.000	5.408.000	8.954.000	83	104	121
JOIN-6 ⋈ store	Junção (JOIN-7)	3.652	11.648	21.538	78	85	78
# Tuplas Resultantes					67	69	64

Tabela B.6: Cardinalidades e resultados para o plano otimizado da Consulta 26

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ date_dim	Filtro de predicados (PF-1)	-	-	-	365	365	365
PF-1 ⋈ store_sales	Junção (JOIN-1)	21.023.610.180	42.049.248.300	63.072.259.515	11.011.824	22.027.618	33.036.315
JOIN-1 ⋈ customer_demographics	Junção (JOIN-2)	21.151.511.539.200	42.310.648.654.400	63.456.153.852.000	10.752.207	21.508.310	32.257.198
σ JOIN-2	Filtro de predicados (PF-2)	-	-	-	89.805	180.270	270.331
PF-2 ⋈ customer_address	Junção (JOIN-3)	11.944.065.000	54.081.000.000	125.974.246.000	89.216	179.123	268.670
σ JOIN-3	Filtro de predicados (PF-3)	-	-	-	13.800	27.603	41.442
PF-3 ⋈ store	Junção (JOIN-4)	607.200	3.091.536	7.376.676	13.766	27.532	41.348
# Tuplas Resultantes					1	1	1

Tabela B.7: Cardinalidades e resultados para o plano otimizado da Consulta 27

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ date_dim	Filtro de predicados (PF-1)	-	-	-	91	91	91
PF-1 ⋈ catalog_sales	Junção (JOIN-1)	2.621.126.053	5.241.236.819	7.862.561.434	823.278	1.647.534	2.473.396
JOIN-1 ⋈ customer	Junção (JOIN-2)	218.991.948.000	988.520.400.000	2.307.678.468.000	821.269	1.643.299	2.467.248
JOIN-2 ⋈ customer_address	Junção (JOIN-3)	109.228.777.000	492.989.700.000	1.149.737.568.000	821.269	1.643.299	2.467.248
σ JOIN-3	Filtro de predicados (PF-2)	-	-	-	66.683	133.090	194.648
# Tuplas Resultantes					586	768	862

Tabela B.8: Cardinalidades e resultados para o plano otimizado da Consulta 29

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ customer_demographics	Filtro de predicados (PF-1)	-	-	-	27.440	27.440	27.440
PF-1 ⋈ store_sales	Junção (JOIN-1)	1.580.514.694.080	3.161.181.844.800	4.741.651.509.840	783.626	1.571.090	2.356.339
σ date_dim	Filtro de predicados (PF-2)	-	-	-	366	366	366
JOIN-1 ⋈ PF-2	Junção (JOIN-2)	286.807.116	575.018.940	862.439.838	153.902	308.760	465.131
σ promotion	Filtro de predicados (PF-3)	-	-	-	354	465	575
JOIN-2 ⋈ PF-3	Junção (JOIN-3)	54.481.308	143.573.400	267.450.325	151.605	304.429	457.951
JOIN-3 ⋈ item	Junção (JOIN-4)	4.244.940.000	15.830.308.000	33.888.374.000	151.605	304.429	457.951
# Tuplas Resultantes					14.000	26.001	37.001

Tabela B.9: Cardinalidades e resultados para o plano otimizado da Consulta 48

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ customer_demographics	Filtro de predicados (PF-1)	-	-	-	27.440	27.440	27.440
PF-1 ⋈ catalog_sales	Junção (JOIN-1)	790.370.317.520	1.580.434.486.960	2.370.864.678.560	411.494	819.873	1.226.105
σ date_dim	Filtro de predicados (PF-2)	-	-	-	365	365	365
JOIN-1 ⋈ PF-2	Junção (JOIN-2)	150.195.310	299.253.645	447.528.325	81.438	164.819	246.088
σ promotion	Filtro de predicados (PF-3)	-	-	-	354	465	575
JOIN-2 ⋈ PF-3	Junção (JOIN-3)	28.829.052	76.640.835	141.500.600	81.119	164.256	244.897
JOIN-3 ⋈ item	Junção (JOIN-4)	2.271.332.000	8.541.312.000	18.122.378.000	81.119	164.256	244.897
# Tuplas Resultantes					13.952	25.954	36.958

Tabela B.10: Cardinalidades e resultados para o plano otimizado da Consulta 91

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
customer_address ⋈ customer	Junção (JOIN-1)	35.378.000.000	1.152.480.000.000	434.778.000.000	266.000	600.000	933.000
σ item	Filtro de predicados (PF-1)	-	-	-	503	935	1329
store_sales ⋈ PF-1	Junção (JOIN-2)	28.972.262.796	107.715.197.700	229.652.144.919	1.020.197	2.040.199	3.039.323
σ date_dim	Filtro de predicados (PF-2)	-	-	-	30	30	30
JOIN-2 ⋈ PF-2	Junção (JOIN-3)	30.605.910	61.205.970	91.179.690	30.134	61.655	93.469
JOIN-1 ⋈ JOIN-3	Junção (JOIN-4)	8.015.644.000	36.993.000.000	87.206.577.000	29.435	60.234	91.259
JOIN-4 ⋈ store	Junção (JOIN-5)	1.295.140	6.746.208	16.244.102	29.092	59.520	90.206
σ JOIN-5	Filtro de predicados (PF-3)	-	-	-	27.791	58.443	89.181
# Tuplas Resultantes					241	458	652

Tabela B.11: Cardinalidades e resultados para o plano otimizado da Consulta 96

Tabelas	Ação	Cardinalidade			Resultado		
		Base 1	Base 2	Base 3	Base 1	Base 2	Base 3
σ d2	Filtro de predicados (PF-1)	-	-	-	214	214	214
PF-1 ⋈ store_returns	Junção (JOIN-1)	1.231.590.330	2.464.134.030	3.696.086.448	521.857	1.042.951	1.564.451
JOIN-1 ⋈ catalog_sales	Junção (JOIN-2)	15.031.351.413.631	60.069.815.182.559	135.171.341.736.074	66.812	140.560	223.026
σ d3	Filtro de predicados (PF-2)	-	-	-	214	214	214
JOIN-2 ⋈ PF-2	Junção (JOIN-3)	14.297.768	30.079.840	47.727.564	4.983	10.139	16.280
JOIN-3 ⋈ store_sales	Junção (JOIN-4)	287.015.478.156	1.168.047.475.380	2.813.195.575.080	485	649	964
σ d1	Filtro de predicados (PF-3)	-	-	-	30	30	30
JOIN-4 ⋈ PF-3	Junção (JOIN-5)	14.550	19.470	28.920	43	45	68
JOIN-5 ⋈ store	Junção (JOIN-6)	1.892	5.040	12.104	38	30	47
JOIN-6 ⋈ item	Junção (JOIN-7)	1.064.000	1.560.000	3.478.000	38	30	47
# Tuplas Resultantes					35	27	44