Explorando arquiteturas multi-core para processamento eficiente de consultas em sistemas de gerência de Big Data

Frank W. R. da Silva[†,¥], Victor T. de Almeida[†,§], Vanessa Braganholo[†]
[†]Instituto de Computação, Universidade Federal Fluminense (UFF)
[¥]Universidade do Estado de Mato Grosso (UNEMAT)
[§]Petrobras S.A.
{frankwrs, valmeida, vanessa}@ic.uff.br

Abstract. *Big Data Management Systems usually manage each machine as one node in parallel query processing pipeline. In multi-core architectures, they leave several processor cores aside that could contribute to speed-up query processing. In this context, this paper explores the use of all available processor cores, assessing the query processing performance in several scenarios. In particular, we use the concept of worker nodes (which are allocated in cores without disk access) and data nodes (which are allocated in cores with disk access) in the same machine using the MyriaX engine as a base platform that supports this concept. We evaluate several cluster configurations varying the amount of data and worker nodes to process two types of queries (self-join and triangle) in a Twitter dataset. The results show that increasing the I/O parallelism in terms of data nodes is not always the most effective strategy. This reinforces the idea of using worker nodes in the query processing pipeline. In the best scenario, we achieved a speed-up of 2.92 by simply adding worker nodes in the available processing cores.*

Resumo. *Sistemas de Gerência de Big Data, em geral, gerenciam cada máquina como um nó dentro do pipeline de processamento paralelo de consultas, deixando de lado núcleos de processador que poderiam contribuir para acelerar o processamento das consultas. Neste contexto, este artigo explora o uso de todos os núcleos de processador disponíveis, avaliando o desempenho de consultas em diversos cenários. Para isso, usamos o conceito de worker nodes (alocados a núcleos que não possuem acesso a disco) e data nodes (alocados em núcleos com acesso a disco) em uma mesma máquina, tendo como plataforma base o mecanismo MyriaX, que suporta este conceito. Avaliamos diversas configurações variando a quantidade de worker nodes e data nodes para dois tipos de consultas (auto-junção e triângulos) em dados do Twitter. Os resultados mostram que aumentar o paralelismo de I/O em termos de data nodes nem sempre é a estratégia mais eficaz, o que reforça a ideia da utilização de worker nodes no pipeline de processamento de consultas. No melhor caso, obtivemos aceleração de 2,92x com a simples adição de worker nodes em núcleos de processamento disponíveis.*

1. Introdução

Sistemas de Gerência de Big Data têm sido usados para processamento de consultas analíticas [DeWitt and Gray, 1992]. Estes sistemas, em sua maioria, utilizam *clusters* para processamento massivamente paralelo de consultas. As arquiteturas utilizadas por eles variam. Alguns exigem que cada máquina tenha um disco acoplado [Abouzeid et al.,

2009; Alsubaiee et al., 2012; Bittorf et al., 2015; Das et al., 2013; Isard et al., 2007; Malewicz et al., 2010; Wang et al., 2017]. Outros utilizam de uma base de dados compartilhada que todos os nós acessam de forma concorrente [Dageville et al., 2016]. Há também sistemas que exploram a tecnologia *multi-core,* alocando vários nós em uma mesma máquina, mas com o recurso de armazenamento compartilhado ou fatiado entre os nós [Gupta et al., 2015; Warneke and Kao, 2009]. No entanto, nenhuma destas estratégias explora totalmente os recursos disponíveis de armazenamento e processamento de forma independente em uma mesma máquina.

Com a finalidade de melhorar a eficiência do processamento paralelo de consultas em sistemas de gerência de Big Data, este artigo explora todos os recursos de processamento (núcleos dos processadores) e armazenamento disponíveis por máquina. Isto 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.

Para avaliar o impacto disso no desempenho de consultas, utilizamos o mecanismo de execução MyriaX, componente do sistema de gerência de Big Data Myria [Wang et al., 2017]. Esta escolha foi motivada pelo fato desse sistema permitir o uso de *worker nodes* e *data nodes* de forma independente. Deste modo, o MyriaX viabiliza a alocação de *data nodes* e *worker nodes* numa mesma máquina, apesar disto nunca ter sido explorado pelo Myria. Em nossa avaliação experimental, utilizamos duas consultas custosas em diversos cenários diferentes. Os resultados mostram que aumentar o paralelismo de I/O em termos de *data nodes* nem sempre é a estratégia mais eficaz, o que reforça a ideia da utilização de *worker nodes* no *pipeline* de processamento de consultas. No melhor caso, obtivemos aceleração de 2,92x pela simples adição de *worker nodes* em núcleos de processamento disponíveis. Além disso, mostramos também que a *cache* tem pouca influência no tempo de processamento das consultas.

As seções a seguir apresentam os trabalhos relacionados (Seção 2), uma visão geral do mecanismo MyriaX (Seção 3), o plano e execução dos experimentos (Seção 4), a avaliação dos resultados (Seção 5) e a conclusão (Seção 6).

2. Trabalhos Relacionados

Clusters de máquinas de propósito geral são utilizados por sistemas de gerência de Big Data, em sua maioria, para processamento massivamente paralelo de consultas. Alguns exigem que cada máquina tenha um disco acoplado, tais como ElasTras [Das et al., 2013], Impala [Bittorf et al., 2015], HadoopDB [Abouzeid et al., 2009], Apache Spark[1], Pregel [Malewicz et al., 2010], Dryad [Isard et al., 2007], Asterix [Alsubaiee et al., 2012], Presto[2] e Vertica[3]. Já o SnowFlake [Dageville et al., 2016] utiliza-se 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 and Kao, 2009], Amazon Redshift [Gupta et al., 2015] e Greenplum[4]. Contudo, em tais sistemas o recurso de armazenamento é fatiado ou compartilhado entre os nós, o que implica em concorrência no acesso ao disco.

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

O serviço Myria fornece um *middleware* compatível com diversos mecanismos de execução de consulta, dentre eles MyriaX, Radish, SciDB, Spark e SPARQL [Wang et al., 2017]. O MyriaX se destaca por utilizar o conceito de *worker nodes* e *data nodes*. D*ata nodes* possuem acesso a disco, enquanto *worker nodes* apenas participam do pipeline de execução de consultas. É possível direcionar o processamento de consultas para *data nodes* e *worker nodes* específicos. Porém, atualmente, o MyriaX utiliza um único tipo de nó por máquina do cluster, independente de quantos núcleos a máquina tenha disponíveis.

De fato, a maioria destes sistemas trata cada máquina como sendo um único nó. Mesmo que cada máquina possua diversos processadores, cada um com diversos núcleos, e vários discos de dados, estes são normalmente gerenciados pelo sistema como um único recurso. No entanto, sabe-se que o uso de mais núcleos implica num melhor potencial de escalabilidade para operadores relacionais, tais como junções [Kim et al., 2009]. Neste contexto, 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. De fato, as estratégias existentes são incapazes de explorar núcleos ociosos de processadores e discos de dados adicionais para atuar no processamento de operadores da consulta.

3. Mecanismo de Execução de Consulta MyriaX

O MyriaX é um mecanismo de execução de consulta relacional *shared-nothing* e paralelo [Wang et al., 2017]. Sua arquitetura é composta por um *master node*, *worker nodes* e *data nodes* [Halperin et al., 2014]. O *master node* realiza o controle dos demais nós. 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 disto nunca ter sido explorado pelo Myria A Figura 1 ilustra uma implantação deste mecanismo com nove máquinas, sendo quatro *data nodes* e quatro *worker nodes*, além do *master node*.

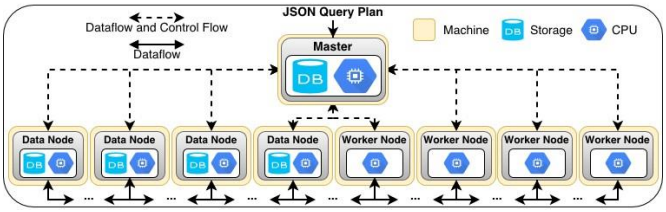

Figura 1. Arquitetura MyriaX com nove máquinas

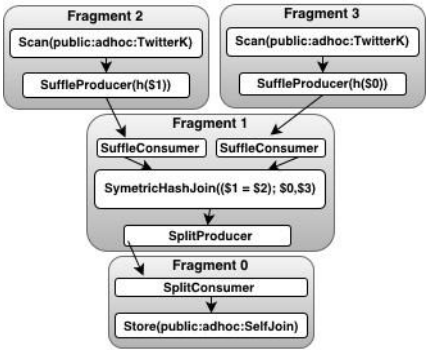

Figura 2. Plano de consulta de auto-junção

No MyriaX, cada máquina que participa do processamento de consultas tem apenas um tipo de nó (*worker node* ou *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*. É importante notar que também é possível conseguir paralelismo de I/O, desde de que haja mais de uma controladora de disco por máquina.

A Figura 2 apresenta um plano de consulta para processamento paralelo de auto-junção (*self-join*) para o MyriaX [Mishra and Eich, 1992]. 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 com a operação *SuffleProducer(h($1))* e os encaminham para os nós da rede que irão efetuar a junção. A operação de *Scan()* faz 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 através de uma porta específica, preenchendo um *buffer* e disparando a execução da junção, utilizando o algoritmo de *HashJoin* [Schneider and 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 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.

No MyriaX, é possível associar vários nós ao mesmo processador, apesar disso não ser utilizado. Na próxima seção, apresentamos experimentos que exploram o uso de vários *worker nodes* na mesma máquina e comparamos o desempenho com o comportamento padrão do MyriaX.

4. Plano e Execução do Experimento

Como mencionado na introdução, máquinas utilizadas no processamento paralelo de consultas podem conter recursos subutilizados, como núcleos de processadores com e sem discos acoplados, que podem ser utilizados como *worker nodes* e *data nodes* adicionais no processamento de consultas. Com base nisto, submetemos o MyriaX a uma coleção de experimentos em *cluster* aplicando diversas configurações e alocações de *data nodes* e *worker nodes*. O *cluster* utilizado é composto de 42 máquinas, cada uma com 2 processadores Intel Xeon *quadcore* (8 núcles), 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.

Para validar a hipótese de que *"é possível diminuir o tempo de processamento de consultas através da adição de worker nodes em núcleos ociosos de processadores*", realizamos diversos experimentos. Nesses experimentos, utilizamos um algoritmo 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*. O 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*.

Tabela 1. Possíveis cenários para oito máquinas com oito núcleos 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

Cada cenário é composto por uma quantidade de máquinas (*m*), *data nodes* (*dn*) e *worker nodes* (*wn*). A quantidade de *worker nodes*, quando maior que zero, é dividida igualmente para cada máquina do cenário. 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 (nesse 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, pois cada uma conta com apenas uma controladora de disco. Os demais nós são alocados como *worker nodes*. Vale ressaltar que nos experimentos, *data nodes* também atuam como *worker nodes* no processamento da consulta. A Tabela 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 and DeWitt, 1997].

A Figura 1 ilustra o possível cenário *m:8_dn:4_wn:4*. Este cenário oferece um maior particionamento da tabela de dados, pois utiliza 4 *data nodes* e 4 *worker nodes*, armazenando uma menor quantidade de dados em cada *data node*, em relação ao cenário *m:2_dn:2_wn:6*, ilustrado na Figura 3, 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. 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.

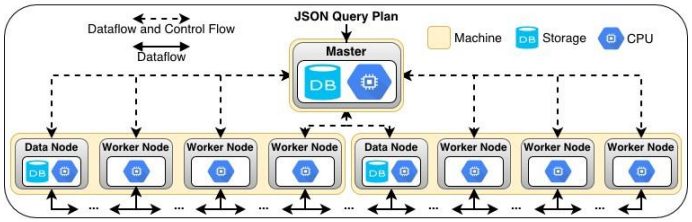

Figura 3. Arquitetura MyriaX com três máquinas

A base de dados utilizada para o experimento foi a do Twitter, 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 tuplas e foi escolhida por ser uma base de dados real com grande volume de dados. Processar certos tipos de consultas (utilizadas nesse experimento) nesse tipo de base é extremamente custoso.

Foram utilizadas duas consultas referentes à base de dados do Twitter. A primeira consulta (C1) realiza auto-junção entre as colunas citadas da base, com a finalidade de identificar relações entre usuários onde ambos seguem um ao outro. O plano desta consulta (Figura 2) foi apresentado e detalhado na Seção 2. A segunda consulta (C2) conta com uma operação de junção a mais e 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 científica quando os dados não cabem na memória [Hu et al., 2013]. Na base de dados utilizada no experimento, a primeira e segunda junções retornam 2.045.216.395 e 89.084.893 tuplas, respectivamente. Durante as submissões das consultas, o algoritmo 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.

5. Resultados e Avaliação

Nos resultados experimentais apresentados nesta seção, cenários padrão do MyriaX estão nomeados como *Baseline* e cenários que se baseiam na hipótese apresentada neste artigo são nomeados *Avaliação*. 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.

Avaliação do Impacto de Memória *Cache*. A Figura 4 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 ganho de aproximadamente 3 segundos, e para a consulta C2 foi *m:4_dn:4_wn:28* com ganho de aproximadamente 15 segundos. 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. Por esse motivo, nos demais experimentos, usamos dados das execuções sem influência de memória *cache*.

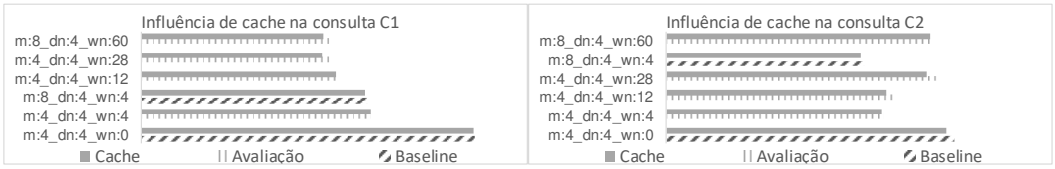

Figura 4. Tempo médio para C1 e C2 com influência de memória *cache*

Avaliação do impacto de *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 5), ou seja, sem particionamento 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. O tempo nesse cenário implicou aceleração de 3,06x para a consulta C1 e 3,44x para a consulta C2, em relação ao cenário *Baseline m:1_dn:1_wn:0*, mesmo sem paralelismo de I/O.

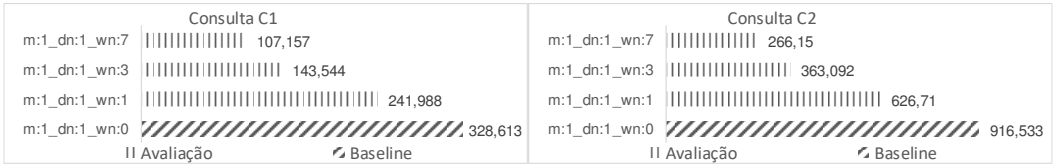

Figura 5. Tempo médio para C1 e C2 referente aos cenários de 1 máquina

Consulta C1 (auto-junção). A Figura 6 apresenta os resultados para a consulta C1 com uso de mais de um *data node*. Logo, há particionamento e paralelismo na leitura de dados. O cenário *Baseline m:2_dn:2_wn:0*, apresentou aceleração de aproximadamente 1,24x referente ao cenário *Baseline m:1_dn:1_wn:0* da Figura 5, que utiliza uma máquina.

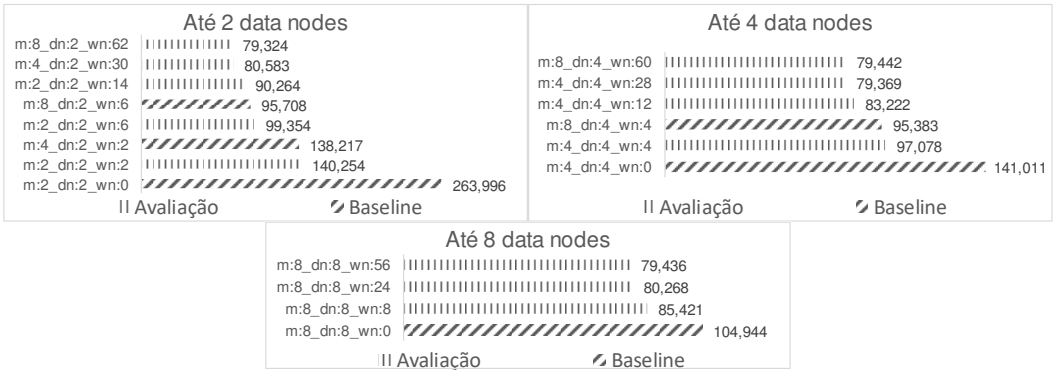

Figura 6. Tempo médio da consulta C1 com até 8 máquinas

Os resultados da Figura 6 com até 2 *data nodes* mostram que a consulta C1 teve melhor desempenho quando submetida em cenários *Avaliação*, com destaque para o cenário *m:8_dn:2_wn:62*, que explora totalmente os recursos disponibilizados utilizando todas as máquinas do cluster. Este cenário apresentou o melhor desempenho com aceleração de aproximadamente 3,33x em relação ao cenário *Baseline m:2_dn:2_wn:0*, que apresentou o maior tempo médio. O resultado deste cenário (*Baseline*) é influenciado pela ausência de *worker nodes* que levam os *data nodes* a executar os demais operadores da consulta de forma sequencial. Por outro lado, os cenários com *worker nodes* iniciam a execução do operador de junção antes da conclusão das operações de leitura de dados, pois *data nodes* encaminham partes dos resultados aos *worker nodes* durante esta operação em *pipeline*. É importante notar que, para cenários com até 2 máquinas, a simples adição de *worker nodes* implicou em aceleração de até 2,92x quando se explora todos os núcleos de processamento disponíveis. Os cenários *Avaliação m:8_dn:2_wn:62* e *m:4_dn:2_wn:30* apresentam ainda menor tempo médio, isto porque aumenta o paralelismo de I/O e os *data nodes* passam a processar menores quantidades de dados.

Nos cenários com até 4 *data nodes* da Figura 6, os cenários *Avaliação* apresentam melhor desempenho em relação aos cenários *Baseline*. Os cenários com melhor desempenho, *m:4_dn:4_wn:28* e *m:8_dn:4_wn:60*, apresentaram aceleração de aproximadamente 1,77x em relação ao cenário *Baseline m:4_dn:4_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, onde aquele que não tem *worker nodes* (*m:4_dn:4_wn:0*) utilizou aproximadamente 3s a mais, justificado pela execução sequencial dos operadores da consulta. 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 1,77x quando se explora todo o recurso de núcleos de processamento disponíveis.

Nos cenários com até 8 *data nodes* da Figura 6, os cenários *Avaliação* também apresentam melhor desempenho em relação aos *Baseline*, porém de forma menos acentuada. O cenário *m:8_dn:8_wn:56*, que explora todo o recurso de processamento e armazenamento disponível nas 8 máquinas, apresenta aceleração de aproximadamente 1,32x 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. Os outros dois cenários *Baseline* (*m:8_dn:2_wn:6* e *m:8_dn:4_wn:4*) apresentam resultados semelhantes pois ambos contêm *worker nodes*, ou seja, realizam *pipeline* no processamento dos operadores da consulta, e também sofrem influência do tráfego de rede pelo motivo de cada nó estar alocado em uma máquina.

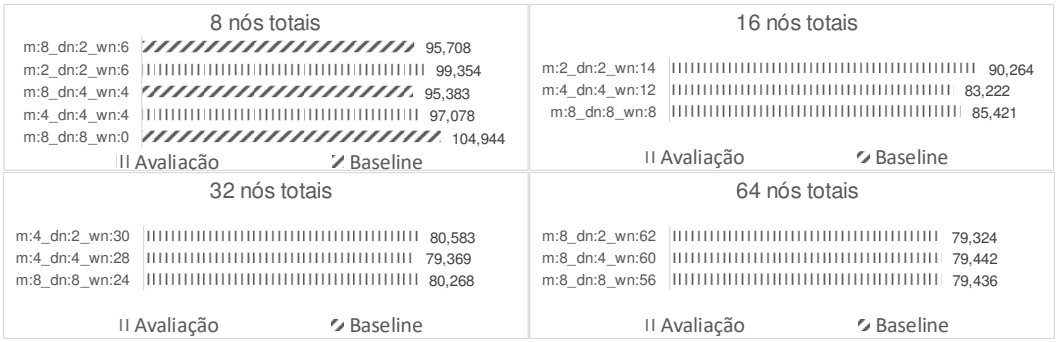

Figura 7. Tempo médio da consulta C1 em cenários com até 8 máquinas agrupados pela quantidade total de nós

A Figura 7 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. Desta forma, a Figura 7 evidencia que, para a consulta C1, é possível atingir um determinado desempenho com menor quantidade de máquinas dimensionando *workers 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.

Consulta C2 (triângulos). A Figura 8 apresenta o tempo médio de 5 rodadas da consulta C2. Em cenários com 64 nós totais, foi realizada apenas uma rodada da consulta, pois o MyriaX apresentou instabilidade no funcionamento.

Os cenários da Figura 8 com até dois *data nodes* mostram que cenários *Avaliação* apresentaram menor tempo médio em relação ao cenário *Baseline*. O cenário *Baseline m:2_dn:2_wn:0* apresentou desaceleração de aproximadamente 1,88x em relação ao cenário *Avaliação* de menor tempo médio (*m:8_dn:2_wn:62)*. Tal disparidade de tempos é influenciada pela execução sequencial dos operadores da consulta C2 realizada pelo cenário *Baseline*. Por outro lado, os cenários que utilizam *worker nodes* apresentam menor tempo médio. Cabe notar que o aumento de *worker nodes* em cenários *Avaliação* com 2 máquinas apresentou aceleração de até 1,7x em relação ao cenário *m:2_dn:2_wn:6*, 1,5x em relação ao cenário *m:2_dn:2_wn:14*. 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. Os cenários *Avaliação m:4_dn:2_wn:30* e *m:8_dn:2_wn:62*, por outro lado, apresentam aceleração de 1,77x e 1,88x, apesar de utilizarem mais *worker nodes*. Tal redução é justificada pelo aumento do paralelismo de I/O com a adição de máquinas no processamento da consulta.

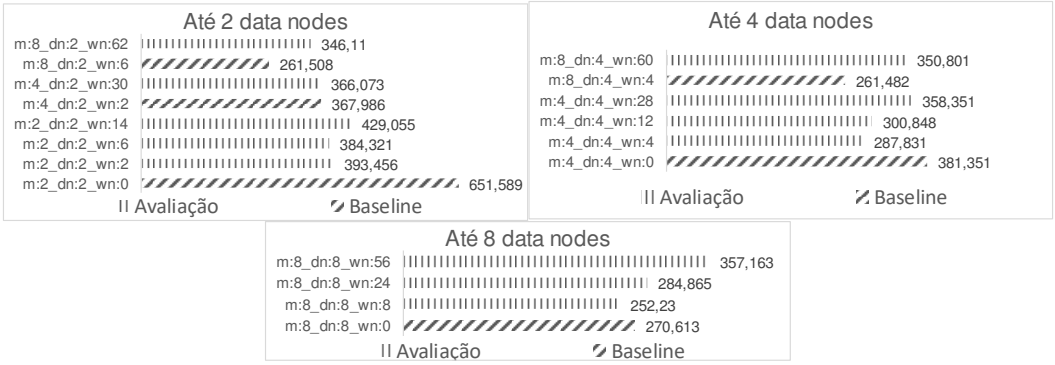

Figura 8. Tempo médio da consulta C2

Nos cenários com até 4 *data nodes* da Figura 8, 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 1,32x em relação ao melhor cenário *Baseline* (*m:4_dn:4_wn:0*). 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. Logo, os cenário *m:4_dn:4_wn:28* e *m:8_dn:4_wn:60*, que exploram todo o recurso disponível de 4 e 8 máquinas, apresentaram desaceleração de aproximadamente 0,8x em relação ao cenário *m:4_dn:4_wn:4*.

Nos cenários com até 8 *data nodes* da Figura 8, os cenários *Baseline* com uso de *worker nodes* também apresentam melhor desempenho em relação a cenários que contêm apenas *data nodes*. O cenário *Avaliação m:8_dn:8_wn:8* apresentou aceleração de até 1,07x e, portanto, o melhor desempenho para as 8 máquinas. É importante notar que o cenário *Avaliação m:8_dn:8_wn:56* apresentou maior tempo médio (desaceleração de 0,76x) no processamento da consulta C2 em relação ao cenário *Baseline* (*m:8_dn:8_wn:0*) para esta quantidade de máquinas. Isto porque, como dito anteriormente, sofre maior influência do tráfego de rede por explorar todos os núcleos de processamento com *worker nodes* em 8 máquinas. Tal desaceleração causada por gargalos de interferência e comunicação já era esperada, como discutido por StoneBraker [1986].

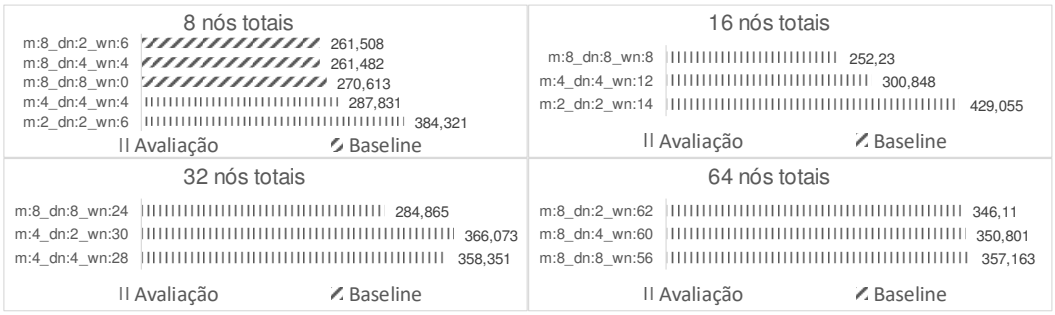

Figura 9. Tempo médio da consulta C2 referente aos cenários com até 8 máquinas agrupados pela quantidade total de nós

A Figura 9 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). Note que cenários

com 64 nós totais apresentam resultados de apenas uma rodada de consulta. O cenário *Avaliação m:8_dn:8_wn:8* apresentou ganho de aproximadamente 9s de tempo de processamento em relação ao cenário *m:8_dn:2_wn:6*, que apresentou menor tempo médio entre os *Baseline*. 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. 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 1,45x 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 9 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* e *m:4_dn:4_wn:4*, apresentaram os melhores desempenhos para a consulta C2 com aceleração de até 1,07x.

6. Conclusão

Sistemas de Gerência de Big Data têm sido usados para processamento massivo paralelo de consulta analíticas. Tais sistemas, em sua maioria, gerenciam cada máquina como um único tipo de nó dentro do *pipeline* de processamento paralelo da consulta. Com o advento da tecnologia *multi-core*, cada máquina passou a poder realizar inúmeras tarefas em paralelo. Neste contexto, este artigo avalia 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 não-compartilhado 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*. Os resultados mostram que, para ambas 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 neste trabalho confirmar a hipótese de pesquisa. 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 desenfreada de *worker nodes* apresentou melhores resultados; em quase todos os casos há uma deterioração a partir de um determinado ponto, de forma bem diferente para as consultas C1 e C2, devido às suas diferentes características. 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 adição de *worker nodes*. Além disso, os resultados mostram que a memória *cache* teve pouca influência no tempo de processamento das consultas. Em trabalhos futuros, pretendemos explorar esta questão, avaliando o comportamento de diferentes

cenários de *data nodes* e *worker nodes* com a aplicação de diversas consultas diferentes agrupadas em cargas de trabalho (*workloads*), o que nos permitirá também avaliar a influência de replicação de dados nos resultados obtidos.

Agradecimentos. Os autores agradecem ao CNPq e a UNEMAT pelo financiamento parcial desse trabalho.

Referências

Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A. and Rasin, A. (2009). 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.

Alsubaiee, S., Behm, A., Grover, R., et al. (2012). ASTERIX: Scalable Warehouse-Style Web Data Integration. *International Workshop on Information Integration on the Web*, p. 1–4.

Bittorf, M., Bobrovytsky, T., Erickson, C. C. A. C. J., et al. (2015). Impala: A Modern, Open-Source SQL Engine for Hadoop. In *Conference on Innovative Data Systems Research (CIDR)*.

Dageville, B., Cruanes, T., Zukowski, M., et al. (2016). The Snowflake Elastic Data Warehouse. *International Conference on Management of Data (SIGMOD)*, p. 215–226.

Das, S., Agrawal, D. and El Abbadi, A. (2013). ElasTraS: An Elastic, Scalable, and Self-Managing Transactional Database for the Cloud. *Transactions on Database Systems (TODS)*, v. 38, n. 1, p. 5.

DeWitt, D. and Gray, J. (1992). Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM*, v. 35, n. 6, p. 85–98.

Gupta, A., Agarwal, D., Tan, D., et al. (2015). Amazon Redshift and the Case for Simpler Data Warehouses. In *International Conference on Management of Data (SIGMOD)*.

Halperin, D., Teixeira de Almeida, V., Choo, L. L., et al. (2014). Demonstration of the Myria Big Data Management Service. *International Conference on Management of Data (SIGMOD)*, p. 881–884.

Hu, X., Tao, Y. and Chung, C.-W. (2013). Massive Graph Triangulation. In *SIGMOD*.

Isard, M., Budiu, M., Yu, Y., Birrell, A. and Fetterly, D. (2007). Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *European Conference on Computer Systems (EuroSys)*.

Kim, C., Kaldewey, T., Lee, V. W., et al. (2009). 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.

Malewicz, G., Austern, M. H., Bik, A. J., et al. (2010). Pregel: A System for Large-scale Graph Processing. In *International Conference on Management of Data (SIGMOD)*.

Mehta, M. and DeWitt, D. J. (1997). Data Placement in Shared-nothing Parallel Database Systems. *The VLDB Journal*, v. 6, n. 1, p. 53–72.

Mishra, P. and Eich, M. H. (1992). Join Processing in Relational Databases. *ACM Computing Surveys (CSUR)*, v. 24, n. 1, p. 63–113.

Schneider, D. A. and DeWitt, D. J. (1989). 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.

Stonebraker, M. (1986). The Case for Shared Nothing. *IEEE Database Engineering*, v. 9, n. 1, p. 4–9.

Wang, J., Baker, T., Balazinska, M., et al. (2017). The Myria Big Data Management and Analytics System and Cloud Service. In *Conference on Innovative Data Systems Research (CIDR)*.

Warneke, D. and Kao, O. (2009). Nephele: Efficient Parallel Data Processing in the Cloud. In *Many-Task Computing on Grids and Supercomputers (MTAGS)*.