paper:9

Uso de SGBDs NoSQL na Gerência da Proveniência Distribuída em *Workflows* Científicos

Guilherme R. Ferreira, Carlos Filipe Jr. e Daniel de Oliveira

Instituto de Computação – Universidade Federal Fluminense (UFF)
{guilherme_rangel, cfmtjunior}@id.uff.br, danielcmo@ic.uff.br

Resumo. *Um fator fundamental na gerência de experimentos modelados como workflows científicos são seus dados de proveniência. Esses dados basicamente são usados para garantir a reprodutibilidade, porém nos últimos anos eles também vêm sendo usados para tarefas de monitoramento e escalonamento de atividades. Como essas tarefas demandam consultas em tempo real, conforme a quantidade de dados de proveniência aumenta, mecanismos eficazes para armazenamento e consulta se fazem necessários. Uma das opções mais comuns é utilizar os SGBDs relacionais para gerenciar a proveniência, dada a tradição da tecnologia. Porém, novas tecnologias como os SGBDs NoSQL tem ganhado bastante atenção nos últimos anos e podem ser de grande valia nesse cenário, principalmente em ambientes distribuídos onde escalabilidade é essencial. Este artigo realiza um estudo comparativo entre SGBDs relacionais e um SGBD NoSQL (Cassandra) no que tange a gerência dos dados de proveniência. Apresentamos um estudo com um workflow real de bioinformática usando a máquina de workflows para nuvens SciCumulus.*

1. Introdução

Os experimentos científicos com base em simulações computacionais são comumente compostos por várias invocações de programas em sequência, cada uma das quais com seu próprio conjunto de parâmetros e dados de entrada (Mattoso *et al.* 2010). Esses experimentos podem ser modelados como *workflows* científicos e gerenciados pelos chamados Sistemas de Gerência de *Workflows* Científicos (SGWfC). Como esses *workflows* são comumente computacionalmente intensivos e processam uma grande quantidade de dados, eles requerem capacidades de paralelismo e uso de ambientes de processamento de alto desempenho (PAD). Vários SGWfCs apresentam mecanismos de execução paralela em ambientes de PAD (Ogasawara *et al.* 2013, Oliveira *et al.* 2010, Taylor *et al.* 2007) como *clusters* (Dantas 2005), grades (Foster e Kesselman 2004) ou nuvens (Vaquero *et al.* 2009). Além de gerenciarem a execução dos *workflows*, esses SGWfCs coletam informação fundamental para o experimento: os dados de proveniência (Freire *et al.* 2008). Os dados de proveniência representam a história e os caminhos dos dados de entrada, desde o início do experimento até o seu fim, e são fundamentais para garantir a sua consequente reprodução e validação (Freire *et al.* 2008).

[1] O trabalho apresentado nesse artigo foi parcialmente financiado pelo CNPq (Processo 478878/2013-3) e FAPERJ (Processo E-26/111.370/2013)

Nos últimos anos, o uso da proveniência tem sido extrapolado para outras áreas como o monitoramento (Pintas *et al.* 2013), a tolerância a falhas (Costa et al. 2012) e o escalonamento adaptativo de atividades (Oliveira *et al.* 2012). Assim, os SGWfCs que são orientados a proveniência como o SciCumulus (Oliveira *et al.* 2010) e o Chiron (Ogasawara *et al.* 2013) agora necessitam realizar consultas frequentes ao repositório de proveniência durante a execução do *workflow* seja para monitora-lo ou para planejar próximas ações. Diferentemente da análise realizada após a execução do *workflow* (*i.e. post mortem*) a qual não necessariamente preza pela eficiência, as análises realizadas em tempo de execução tendem a requerer uma maior velocidade tanto para armazenar os dados quanto para consultá-los de forma que não atrapalhe a execução paralela do *workflow*. Dessa forma, conforme a escala do *workflow* aumenta (seja na quantidade de atividades ou na quantidade de dados processados) o volume de dados de proveniência gerenciados também aumenta proporcionalmente e mecanismos eficientes de armazenamento e consulta a esses dados se fazem necessários de forma que não interfiram no desempenho da máquina de *workflows*. Mais ainda, *workflows* executados em ambientes distribuídos como as nuvens de computadores podem aumentar a quantidade de recursos computacionais usados na execução do *workflow*, aumentando assim a quantidade de dados de proveniência a serem gerenciados o que faz com que a abordagem escolhida para gerenciar os dados de proveniência seja capaz de aumentar a sua capacidade também.

De fato, esse problema da gerência dos dados de proveniência se assemelha muito aos problemas de *Big Data* (Bertino et al. 2012). O termo *Big Data* é usado para descrever qualquer coleção de conjuntos de dados que são grandes e complexos que se torna difícil trabalhar com esses dados usando as técnicas de processamento de dados tradicionais. E principalmente, em problemas de *Big Data* o aumento e diminuição da demanda em tempo real também é um problema relevante e ainda sem solução definitiva. Assim, podemos considerar a gerência da proveniência distribuída como um desses problemas de *Big Data* já que o volume de dados pode ser grande e a demanda por armazenamento e consulta pode aumentar e diminuir ao longo do tempo. Por exemplo, se considerarmos uma base de dados de proveniência que siga o modelo PROV-Wf (Costa *et al.* 2013, Oliveira *et al.* 2014), para uma base que contenha 10.000 execuções de *workflows*, teremos em média mais de 100.0000 registros de atividades, mais de 1.000.000 de registros de execuções de atividades e mais de 5.000.000 de registros de arquivos gerados. Em relação ao espaço em disco ocupado, essa base não é extremamente grande (12GB), porém como possui muitos registros, as consultas executadas pela máquina de execução podem ser demoradas, o que também caracteriza um problema de *Big Data*.

Atualmente, muitos dos SGWfCs optam por armazenar os dados de proveniência em bancos de dados relacionais. Essa é uma solução natural, uma vez que a tecnologia presente nos SGBDs relacionais está bem sedimentada e estável. Os SGBDs relacionais atuais são bastante eficientes, porém eles possuem a desvantagem de não escalarem facilmente. Ou seja, se a demanda por armazenamento e consulta aumentar rapidamente (*i.e. flash crowd*) não é trivial aumentar a capacidade do SGBD tão rapidamente. Em paralelo, novos SGBDs não-relacionais vem sendo propostos como soluções escaláveis e "vendidos" como soluções ideais principalmente para os problemas de *Big Data*. Os SGBDs não relacionais (a partir de agora chamados somente de NoSQL) não garantem as tradicionais propriedades ACID dos SGBDs relacionais, porém focam em prover a

chamada escalabilidade horizontal (aumento ou diminuição da capacidade de acordo com a demanda). Os SGBDs NoSQL passam a ser uma alternativa atraente para armazenar a proveniência de *workflows* executados em ambientes distribuídos.

Assim, esse artigo apresenta um estudo preliminar sobre o uso de SGBDs NoSQL frente aos tradicionais SGBDs relacionais no que tange a gerência dos dados de proveniência desses *workflows*. O principal aspecto a ser analisado nesse artigo é o impacto no SGWfC quando o tipo de SGBD utilizado é modificado. Apresentamos a migração dos dados de proveniência de um banco de dados no PostgreSQL para o SGBD NoSQL Cassandra, que possui a vantagem de ser mais escalável e eficiente para grandes volumes de dados. O banco de dados de proveniência migrado é o mesmo utilizado pelo SGWfC SciCumulus (Oliveira *et al.* 2010) que executa *workflows* em paralelo em ambientes de nuvem. Além da migração da base de dados para uma base não-relacional, esse artigo também apresenta uma análise comparativa entre os resultados obtidos pela versão anterior no que se refere ao impacto da mudança de tecnologia no desempenho da máquina de *workflow*.

Além da introdução, esse artigo possui outras 4 seções. A Seção 2 apresenta o desenvolvimento e a migração da base de dados de proveniência do PostgreSQL para o Cassandra. A Seção 3 apresenta uma análise dos resultados experimentais. A Seção 4 apresenta os trabalhos relacionados e a Seção 5 conclui este artigo.

2. Mapeamento do Banco de Dados de Proveniência

Nesse artigo, escolhemos o Cassandra como o SGBD NoSQL a ser usado. O Cassandra é um SGBD de natureza distribuída que utiliza o modelo de registros particionados com consistência configurável. Esses registros são armazenados em tabelas chamadas de *Column Families*. Nessas tabelas, a primeira parte da chave primária é chamada chave de partição e serve para definir em qual partição o registro será armazenado. Dentro de uma partição, os registros são agrupados pela chave de agrupamento ou *clustering keys*. Um ponto importante que deve ser ressaltado é a não existência de chaves estrangeiras no modelo do Cassandra. Isso permite que tabelas sejam criadas, removidas ou alteradas em tempo de execução, sem bloqueios para atualizações ou consultas. Desta forma, não existe o conceito de junção e sub-consultas.

O Cassandra distribui os dados de forma transparente, bastando o usuário definir na hora da criação de um *keyspace* em quantos nós (máquinas) os dados devem ser distribuídos e qual política de replicação utilizar. Além disso, o usuário deve definir qual máquina será a *seed* que tem como função armazenar a lista dos nós pertencentes ao conjunto de nós disponíveis para que os outros nós possam encontrá-los. O Cassandra não utiliza SQL como linguagem de consulta. Sendo assim, ele possui a sua própria linguagem de consulta e manipulação de dados chamada CQL (*Cassandra Query Language*). A linguagem possui pontos similares com o SQL, porém possui algumas limitações. Por exemplo, não é possível realizar uma seleção em uma coluna que seja referenciada na cláusula WHERE sem que a mesma faça parte da chave primária ou possua um índice secundário declarado. Ainda em relação à cláusula WHERE, no caso da chave de partição ser composta, sempre que uma de suas partes aparecer na projeção, as outras devem, obrigatoriamente, se fazer presentes.

Existem outras restrições importantes, como por exemplo a restrição do uso dos operadores "=" e "IN" apenas para os campos da chave de partição. Para os outros

campos que não fazem parte da chave de partição, outros operadores lógicos podem ser usados com a exceção de "<>". Existem também os índices secundários, que permitem que a consulta seja realizada sobre algum atributo que não esteja declarado na chave primária, porém quando um atributo que possui um índice secundário aparece em uma consulta, é necessário visitar todos os nós em que a *keyspace* se faz presente, não sendo, portanto, uma consulta otimizada quanto uma realizada apenas sobre os atributos da chave primária. Ou seja, é aconselhável evitar a criação de índices secundários sem que haja real necessidade. Outro ponto de diferença é a questão da ordenação. Apesar de utilizar a cláusula ORDER BY como em SQL, no CQL somente é possível ordenar pelos atributos presentes na chave de agrupamento e, é necessário respeitar sua ordem de declaração.

Dadas as restrições impostas pelo Cassandra, iniciamos o mapeamento do banco de dados proveniência relacional para o modelo do Cassandra. Para tal, escolhemos mapear o banco de dados proveniência do SciCumulus se encontra atualmente no PostgreSQL e é baseado no modelo PROV-Wf (Costa *et al.* 2013) que estende o padrão PROV (Moreau e Missier 2013) do W3C. Logo, é necessário mapear o esquema levando-se em conta as restrições do Cassandra. O modelo utilizado no PostgreSQL foi adaptado de forma que não houvesse perda de semântica. Dessa forma, cada tabela que existia no PostgreSQL foi mapeada para uma *Column Family* no Cassandra, sendo que sempre que houvesse uma restrição como uma das anteriormente citadas, teríamos que adaptar o *script* de criação dos objetos. Devido às restrições de espaço, exemplificaremos apenas uma tabela do sistema, a *hactivation*. Entretanto, o mapeamento completo do banco de dados do SciCumulus para o seu equivalente no Cassandra se encontra disponível em *https://s3.amazonaws.com/SBBD-Cassandra/Mapeamento-PostgreSQL-Cassandra.zip*. A Tabela 1 apresenta o *script* de criação apenas da tabela *hactivation* no PostgreSQL e no Cassandra.

Tabela 1 Mapeamento do *script* de criação da tabela *hactivation*

PostgreSQL:

```
CREATE TABLE hactivation (
  taskid integer NOT NULL DEFAULT,
  actid integer NOT NULL,
  processor integer,
  exitstatus integer,
  commandline text,
  workspace character varying(150),
  terr text,
  tout text,
  starttime timestamp with time zone,
  endtime timestamp with time zone,
  status character varying(25),
  CONSTRAINT htask_pk PRIMARY KEY (actid,
taskid),
  CONSTRAINT htask_actid_fk FOREIGN KEY
(actid)
      REFERENCES hactivity (actid) MATCH
SIMPLE
      ON UPDATE NO ACTION ON DELETE NO
ACTION)
```

Cassandra:

```
CREATE TABLE hactivation (
  id uuid,
  taskid timeuuid,
  actid uuid,
  status text,
  seqnum int,
  processor int,
  exitstatus int,
  commandline text,
  workspace text,
  terr text,
  tout text,
  starttime TIMESTAMP,
  endtime TIMESTAMP,
  extractor text,
  PRIMARY KEY (id, taskid)
);
CREATE INDEX ON hactivation(status);
CREATE INDEX ON hactivation(actid);
CREATE INDEX ON
hactivation(workspace);
CREATE INDEX ON hactivation(seqnum);
```

Na Tabela 1 podemos perceber algumas diferenças importantes. Primeiramente, no Cassandra temos a criação de um UUID (*i.e. Universally Unique Identifer*) que é um

identificador universal que evita que ocorram colisões de identificadores no momento do armazenamento. Basicamente, quando não utilizamos o UUID são realizadas diversas atualizações em uma tabela de metadados para se recuperar o último identificador utilizado de cada tabela, o que acaba impactando no tempo final de execução do *workflow*. Além disso, não é possível executar mais de um *workflow* concorrentemente, pois existe o caso de uma instância do SGWfC consultar a tabela de metadados para resgatar o último identificador e uma outra instância do SGWfC consultar a tabela antes que a anterior pudesse gravar o último identificador utilizado. Nesse caso, teríamos duas instâncias de SGWfC utilizando os mesmos identificadores e existiriam informações conflitantes, e como o SciCumulus é orientado aos dados de proveniência, a execução de um *workflow* acabaria interferindo na outra.

Além disso, índices foram criados para todos os campos que faziam parte de predicados de seleção de alguma consulta executada pelo SciCumulus. Por exemplo, no caso do campo *status*, o SciCumulus verifica quais tarefas já executaram via proveniência e isso requer uma consulta onde *status* = 'FINISHED', logo um índice sobre esse campo se faz necessário no Cassandra. Além disso, antes de criamos as *Column Families* no Cassandra, devemos definir uma *keyspace* a ser usada. No caso do experimento realizado nesse artigo criamos uma *keyspace* específica para as *Columns Families* do SciCumulus conforme apresentado na Tabela 2. Na Tabela 2 podemos perceber que a *keyspace* criada está replicada em um nó utilizando a estratégia *SimpleStrategy* que posiciona a primeira réplica do banco de proveniência em um nó determinado pelo particionador e as réplicas adicionais são colocadas no próximo nó no sentido horário do anel de nós que fazem parte da rede de replicação do Cassandra. Utilizando os bancos de dados relacional e NoSQL executamos então um estudo comparativo entre as abordagens para verificar as vantagens e as desvantagens da utilização de SGBDs NoSQL e seu impacto nas máquinas de execução de *workflows*.

Tabela 2 Criação da *keyspace* no Cassandra

```
CREATE KEYSPACE sciphyComUUID WITH REPLICATION = {'class' :
'SimpleStrategy','replication_factor' : 1 };
USE sciphyComUUID;
```

3. Resultados Experimentais

A fim de comparar o desempenho do SGWfC SciCumulus utilizando o PostgreSQL e o Cassandra, é necessário estabelecer as mesmas condições de execução do SciCumulus para ambos os SGBDs. Para tal, foi configurado o ambiente com a mesma capacidade de processamento tanto para o SciCumulus-PostgreSQL quanto para o SciCumulus-Cassandra. Em ambos os casos utilizamos 16 máquinas virtuais do tipo *small* (EC2 ID: m1.small - 1.7 GB RAM, 1 núcleo de CPU equivalente a um 1.0-1.2 GHz 2007 Opteron, 160 GB de armazenamento) no ambiente da Amazon EC2, um ambiente de nuvem com recursos sob demanda.

Como o SciCumulus é orientado ao banco de proveniência, ele consulta o banco de dados para realizar as várias tarefas como escalonamento e monitoramento. Qualquer mudança de SGBD pode fazer com que a execução paralela do *workflow* também seja impactada negativamente. Para analisar o impacto da mudança do SGBD no SciCumulus, executamos o *workflow* SciPhy (Ocaña *et al.* 2011) utilizando tanto o PostgreSQL quanto o Cassandra. O SciPhy é um *workflow* de bioinformática que executa uma varredura de parâmetros em um conjunto de sequências de DNA, RNA ou

aminoácidos e gera diversas árvore filogenéticas, que apresentam o quanto um organismo se assemelha a outro evolutivamente. Para mais detalhes, favor referenciar Ocaña *et al.* (2011). O SciPhy foi executado com vários conjuntos de dados de entrada de forma que pudéssemos analisar como o SGBD NoSQL se comportaria com o aumento do volume de dados de entrada. A Figura 1 apresenta os tempos de execução do *workflow* para cada conjunto de dados de entrada com apenas uma máquina executando os SGBDs PostgreSQL e o Cassandra (instâncias rotuladas como 1N) e com três máquinas executando os SGBDs Cassandra e o PostgreSQL com o Pgpool (instâncias rotuladas como 3N).

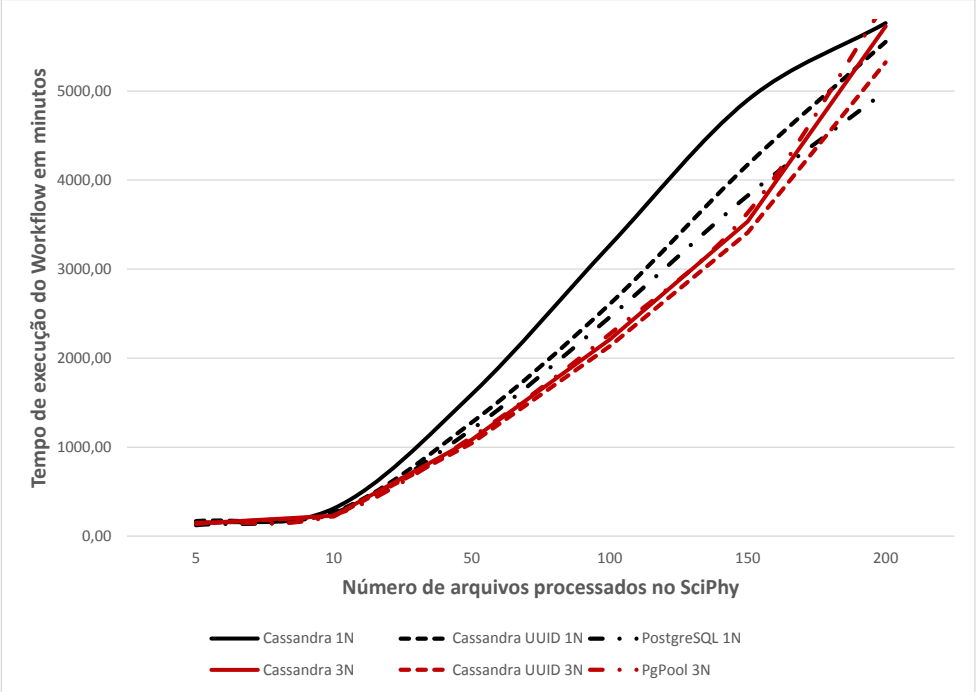

Figura 1 Tempo de execução do *workflow SciPhy*

Nos casos das execuções com os SGBDs em apenas um nó, podemos perceber que sem a distribuição efetiva dos dados o Cassandra insere um *overhead* significativo na execução do *workflow*. Por exemplo, quando processamos 200 arquivos de entrada o tempo de execução do SciCumulus com PostgreSQL dura 4.998 minutos, com o Cassandra sem UUID a duração é de 5.763 minutos e com o Cassandra com UUID a duração é de 5.552 minutos. Apesar de sabermos que a versão sem UUID é menos eficiente já que necessita realizar consultas sucessivas a tabela de metadados, mesmo com a utilização do UUID ainda temos uma degradação de 11,08% no tempo de execução do *workflow*, o que gera uma diferença absoluta de 9,2 horas. Essa diferença ocorre uma vez que o SciCumulus necessita executar diversas consultas aos dados de proveniência durante o curso de execução do *workflow*. Toda vez que uma consulta no PostgreSQL necessita realizar uma junção (*e.g.* saber quais são as atividades de um determinado *workflow*), mais de uma consulta deve ser realizada no Cassandra, o que faz com que a cada atividade a ser despachada, uma consulta seja executada com um *overhead* embutido.

Analisamos também o caso em que o SGBD se encontra distribuído em 3 nós (séries em vermelho na Figura 1). Nesses casos, tanto a execução do Cassandra com UUID quanto a sem o uso do UUID apresentaram um desempenho melhor do que o

PostgreSQL utilizando o Pgpool. Essa vantagem se deu já que cada nó executor do SciCumulus grava seus dados de proveniência o que faz com que tenhamos várias centenas de gravações concorrentes no SGBD. O Cassandra é otimizado justamente para esse fim (acelerar as inserções) o que fez com que seu desempenho tenha sido melhor. O ganho na inserção no Cassandra se deve a associação de um disco em separado para a gravação de *logs*. Quando as *Column Families* compartilham a mesma unidade de disco com o *log*, ocorrem deteriorações nas operações. No caso, a execução com o uso de UUID foi 19% mais eficiente do que com o uso do Pgpool. É importante ressaltar que acima de 175 arquivos de entrada para o SciPhy, o tempo das execuções com o SGBD em 3 nós foi praticamente equivalente a execução com 1 nó para o SGBD. Esse comportamento se deveu a um problema de implementação da máquina de *workflows*. A versão original do SciCumulus elegia apenas uma máquina do seu *cluster* virtual para capturar e armazenar os dados de proveniência (chamada de nó 0). Para os experimentos apresentados nesse artigo, foi necessário modificar a estrutura da máquina para que todos os nós pudessem gravar dados de proveniência. Essa alteração gerou *overheads* que impactaram na execução nos casos com mais instâncias de atividades existentes.

Em todos os casos apresentados anteriormente, executamos apenas uma instância do SciPhy. Para analisar como o Cassandra se comportaria com vários *workflows* executando concorrentemente, iniciamos 4 instâncias do SciPhy consumindo 200 arquivos de entrada cada uma e utilizando o Cassandra com UUID (já que sem o uso do UUID não faria sentido) e o PostgreSQL e Pgpool (Figura 2). O Cassandra apresentou um desempenho superior em todos os casos, mas principalmente quando utilizadas 3 nós para o SGBD. Esse desempenho se dá justamente devido a sua otimização de escrita e também ao controle de acessos concorrentes do Cassandra, que é otimizado. No caso, a execução do Cassandra com 3 nós foi 12% mais eficiente que a execução com o Pgpool o que nos dá uma diferença de aproximadamente 9 horas.

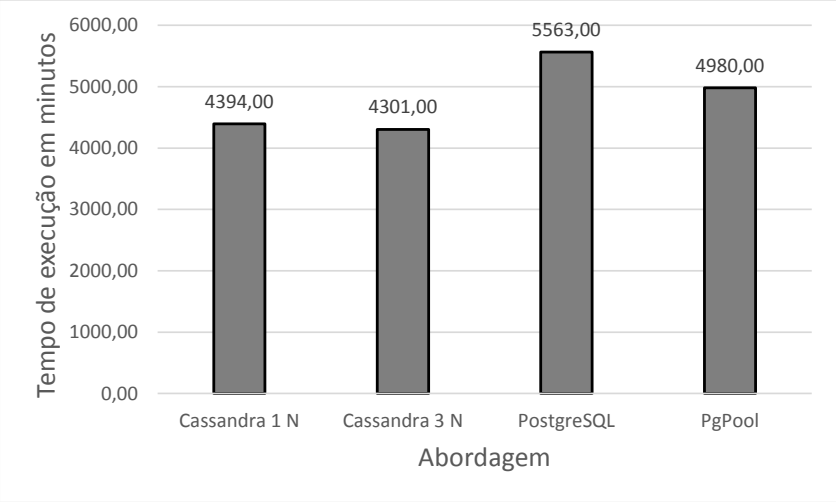

Figura 2 Tempo de execução de 4 *workflows* concorrentes

4. Trabalhos Relacionados

Apesar de existirem diversos trabalhos na literatura que focam na comparação entre SGBDs NoSQL (Abramova e Bernardino 2013, Tudorica e Bucur 2011, Wang e Tang

2012), eles não são específicos para os dados de proveniência. Entretanto, existem alguns trabalhos que comparam o uso de SGBDs relacionais com SGBDs NoSQL especificamente para o armazenamento de dados de proveniência. Cheah (2014) apresenta um estudo comparativo sobre a gerência de dados de proveniência em ambientes de alto desempenho. Similarmente, Vicknair *et al.* (2010) e McColl *et al.* (2014) avaliam o uso de SGBDs NoSQL para o armazenamento e consulta a dados de proveniência distribuída. Entretanto, todos esses trabalhos consideram apenas o uso de SGBDs baseados em grafos como o Neo4J e não baseados em registros distribuídos como o Cassandra e não avaliam o impacto dos SGBDs NoSQL nas máquinas de *workflow*. Desta forma, esse trabalho se diferencia ao explorar uma categoria diferente de SGBDs NoSQL e seu impacto em uma máquina de *workflows* orientada a proveniência.

5. Conclusões

Este artigo apresenta um estudo inicial e preliminar sobre o uso do SGBDs NoSQL para a gerência de dados de proveniência distribuídos e seu consequente impacto nas máquinas de execução de *workflows* científicos. Nesse artigo, escolhemos o Cassandra como SGBD NoSQL e apesar do Cassandra não apresentar todas as vantagens dos tradicionais SGBDs relacionais, o mesmo se mostrou uma solução interessante de acordo com o aumento da quantidade de inserções e acessos concorrentes ao SGBD. Como a captura e a gerência dos dados de proveniência podem ser considerados problemas de *Big Data*, principalmente quando vários *workflows* são executados concorrentemente, o Cassandra pode oferecer um benefício único, já que o mesmo é otimizado para múltiplas escritas e acessos concorrentes a base. Por exemplo, quando o *workflow* SciPhy foi executado com 200 arquivos de entrada o tempo necessário para a execução do *workflow* com o Cassandra em 3 nós foi 12% inferior ao tempo de execução utilizando o Pgpool em 3 nós de mesma capacidade de processamento. Esse resultado é um indicativo do potencial do Cassandra para cenários de múltiplos acessos concorrentes, entretanto, mais experimentos se fazem necessários. Os resultados, apesar de positivos em alguns cenários ainda não podem ser generalizados para qualquer classe de *workflow*. Os resultados obtidos são representativos apenas para uma classe de *workflows*, na qual o SciPhy está incluído. O SciPhy é um *workflow* no estilo MapReduce (Dean e Ghemawat 2010). Novos experimentos devem ser executados, principalmente no que tange as consultas *post mortem* que também são fundamentais para a análise dos resultados do experimento.

Referências Bibliográficas

Abramova, V., Bernardino, J., (2013), "NoSQL Databases: MongoDB vs Cassandra". In: *Proceedings of the International C* Conference on Computer Science and Software Engineering*, p. 14–22, New York, NY, USA.

Bertino, E., Bernstein, P., Agrawal, D., Davidson, S., Dayal, U., Franklin, M., Gehrke, J., Haas, L., Halevy, A., et al., (2012), "Challenges and Opportunities with Big Data"

Cheah, Y.-W., (2014), *Quality, Retrieval and Analysis of Provenance in Large-scale Data*, Indiana University, AAI3613730.

Costa, F., de Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E., Mattoso, M., (2012), "Enabling Re-executions of Parallel Scientific Workflows Using Runtime Provenance Data". In: *Proceedings of the 4th International Conference on Provenance and Annotation of Data and Processes*, p. 229–232, Berlin, Heidelberg.

Costa, F., Silva, V., de Oliveira, D., Ocaña, K., Ogasawara, E., Dias, J., Mattoso, M., (2013), "Capturing and querying workflow runtime provenance with PROV: a practical approach". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, p. 282–289, New York, NY, USA.

Dantas, M., (2005), "Clusters Computacionais", *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*, 1 ed.Rio de Janeiro: Axcel Books, p. 145–180.

Dean, J., Ghemawat, S., (2010), "MapReduce: a flexible data processing tool", *Communications of the ACM*, v. 53, n. 1, p. 72–77.

Foster, I., Kesselman, C., (2004), *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann.

Freire, J., Koop, D., Santos, E., Silva, C. T., (2008), "Provenance for Computational Tasks: A Survey", *Computing in Science and Engineering, v.10*, n. 3, p. 11–21.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Ogasawara, E., Oliveira, D., Cruz, S. M., Martinho, W., Murta, L., (2010), "Towards supporting the life cycle of large scale scientific experiments", *International Journal of Business Process Integration and Management*, v. 5, n. 1, p. 79.

McColl, R. C., Ediger, D., Poovey, J., Campbell, D., Bader, D. A., (2014), "A Performance Evaluation of Open Source Graph Databases". In: *Proceedings of the First Workshop on Parallel Programming for Analytics Applications*, p. 11–18, New York, NY, USA.

Moreau, L., Missier, P., (2013). PROV-DM: The PROV Data Model. Disponível em: http://www.w3.org/TR/2013/REC-prov-dm-20130430/. Acesso em: 17 Feb 2014.

Ocaña, K. A. C. S., Oliveira, D., Ogasawara, E., Dávila, A. M. R., Lima, A. A. B., Mattoso, M., (2011), "SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes". In: *Advances in Bioinformatics and Computational Biology*, p. 66–70, Berlin, Heidelberg.

Ogasawara, E., Dias, J., Silva, V., Chirigati, F., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M., (2013), "Chiron: A Parallel Engine for Algebraic Scientific Workflows", *Concurrency and Computation*, v. 25, n. 16, p. 2327–2341.

Oliveira, D., Ocaña, K., Baião, F., Mattoso, M., (2012), "A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds", *Journal of Grid Computing*, v. 10, n. 3, p. 521–552.

Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M., (2010), "SciCumulus: a lightweight cloud middleware to explore many task computing paradigm in scientific workflows". In: *3rd International Conference on Cloud ComputingInternational Conference on Cloud Computing*, p. 378–385, Washington, DC, USA.

Oliveira, W., Oliveira, D., Braganholo, V., (2014), "Experiencing PROV-Wf for Provenance Interoperability in SWfMSs". In: *IPAW*, Cologne, German.

Pintas, J., Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E., Mattoso, M., (2013), "SciLightning: a Cloud Provenance-based Event Notification for Parallel Workflows". In: *Proceedings of the 3rd International Workshop on Cloud Computing and Scientific Applications (CCSA)3rd International Workshop on Cloud Computing and Scientific Applications (CCSA)*, p. 352–365, Berlin, Germany.

Taylor, I. J., Deelman, E., Gannon, D. B., Shields, M., (2007), *Workflows for e-Science: Scientific Workflows for Grids*. 1 ed. Springer.

Tudorica, B. G., Bucur, C., (2011), "A comparison between several NoSQL databases with comments and notes". In: *Roedunet International Conference (RoEduNet), 2011 10thRoedunet International Conference (RoEduNet), 2011 10th*, p. 1–5

Vaquero, L. M., Rodero-Merino, L., Caceres, J., Lindner, M., (2009), "A break in the clouds: towards a cloud definition", *ACM SIGCOMM Computer Communication Review*, v. 39, n. 1, p. 50–55.

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D., (2010), "A Comparison of a Graph Database and a Relational Database: A Data Provenance Perspective". In: *Proceedings of the 48th Annual Southeast Regional Conference*, p. 42:1–42:6, New York, NY, USA.

Wang, G., Tang, J., (2012), "The NoSQL Principles and Basic Application of Cassandra Model". In: *2012 International Conference on Computer Science Service System (CSSS)2012 International Conference on Computer Science Service System (CSSS)*, p. 1332–1335