Seminário de Andamento
Escalonamento Adaptativo para o Apache Hadoop

Guilherme Weigert Cassales[1], **Andrea Schwertner Charão**[1]

[1]Laboratório de Sistemas Computacionais
Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

{cassales,andrea}@inf.ufsm.br

Abstract. *This article proposes to improve Apache Hadoop scheduling through the usage of context-awareness. Apache Hadoop is the most popular implementation of the MapReduce paradigm for distributed computing, but its design doesn't adapt automatically to computing nodes' context and capabilities. By introducing context-awareness into Hadoop, we intent to dynamically adapt its scheduling to the execution environment. The solution has been incorporated into Hadoop and evaluated through controlled experiments. The experiments demonstrate that context-awareness provides comparative performance gains, especially when part of the resources disappear during execution.*

Resumo. *Este trabalho propõe uma melhoria no escalonamento do Apache Hadoop através da utilização de informações de contexto sobre os nós de um cluster. O Apache Hadoop é a implementação mais popular do paradigma MapReduce em computação distribuída, porém possui alguns problemas de performance em ambientes dinâmicos. Ao introduzir sensibilidade ao contexto no Hadoop, espera-se que o escalonamento das aplicações adapte-se às mudanças do ambiente. A solução foi implementada no Apache Hadoop e testes preliminares indicam que houve uma melhora de performance, especialmente quando parte dos recursos desaparece durante a execução.*

1. Introdução

Apache Hadoop é um *framework* de computação paralela e distribuída para processamento de grandes conjuntos de dados e implementa o paradigma de programação MapReduce [Dean and Ghemawat 2008]. O Apache Hadoop é projetado para ser escalável de um único servidor a milhares de máquinas, cada uma oferecendo processamento e armazenamento local. Esta capacidade de utilizar grande quantidade de hardware barato e a crescente importância do processamento de dados não estruturados tornaram o Apache Hadoop uma ferramenta relevante no mercado [Su and Swart 2012].

Sem uma configuração específica pelo administrador, o Apache Hadoop assume que está sendo utilizado em um *cluster* homogêneo para execução de aplicações MapReduce. Uma vez que o desempenho geral do *cluster* está ligado ao escalonamento de tarefas, o desempenho do Hadoop pode diminuir significativamente quando executado em ambientes que não satisfaçam a suposição feita no projeto do *framework*, ou seja, em ambientes dinâmicos ou heterogêneos [Kumar et al. 2012].

Esta é uma preocupação especial quando tenta-se utilizar o Hadoop em grids pervasivos. Grids pervasivos são uma alternativa aos custosos *clusters* dedicados, dado que a aquisição e manutenção de um *cluster* dedicado continua alta e dissuasiva para muitas organizações. De acordo com [Parashar and Pierson 2010], grids pervasivos representam a generalização extrema do conceito de grid por possuírem recursos pervasivos, ou seja, recursos computacionais ociosos incorporados em ambientes pervasivos são requisitados com objetivo de processar tarefas de maneira distribuída.

Estes grids podem ser vistos como grids formados por recursos existentes (desktops, servidores, etc.) que ocasionalmente contribuem para o poder de processamento do grid. Estes recursos são inerentemente heterogêneos e potencialmente móveis, entrando e saindo do grid dinamicamente. Sabendo disto, é possível afirmar que grids pervasivos são, essencialmente, ambientes heterogêneos, dinâmicos e compartilhados. Ainda, com base nesta afirmação, seu gerenciamento eficiente torna-se um trabalho muito complexo, afetando severamente o escalonamento de tarefas [Nascimento et al. 2008].

Muitos trabalhos propuseram-se a melhorar a adaptabilidade do *framework* Apache Hadoop em ambientes que divergem da suposição inicial, cada um possuindo sua própria proposta e objetivos [Kumar et al. 2012, Zaharia et al. 2008, Rasooli and Down 2012, Sandholm and Lai 2010, Steffenel et al. 2013].

Sabe-se que, o Apache Hadoop é baseado em configuração estática de arquivos e que as versões correntes não adaptam-se a variações de recursos ao longo do tempo. Além disto, os procedimentos de instalação forçam o administrador a definir manualmente as características de cada recurso em potencial, como a memória e o número de cores de cada máquina, tornando a tarefa difícil e demorada em ambientes heterogêneos. Todos estes fatores impedem a utilização da capacidade total do Hadoop em ambientes voláteis, portanto, é essencial possuir sensibilidade ao contexto para tornar esta adaptação possível.

Este trabalho propõe-se a introduzir sensibilidade ao contexto nos mecanismos de escalonamento do Apache Hadoop através da coleta e transmissão de dados. Buscou-se atingir estes objetivos com o mínimo de intrusão e alterações nos mecanismos chave de escalonamento possível.

2. Revisão de Literatura

É necessário definir alguns termos, técnicas e/ou ferramentas para o entendimento do trabalho. Por exemplo, a conceituação formal de sensibilidade ao contexto é muito importante, uma vez que constitue-se da técnica de obtenção dos dados. Como complemento à sensibilidade ao contexto define-se o que é o ZooKeeper, a ferramenta utilizada para transmissão dos dados coletados. Além disso, a compreensão de como o Apache Hadoop e seu escalonamento funcionam, bem como quais trabalhos já foram feitos neste âmbito, são relevantes.

2.1. Sensibilidade ao Contexto

Sensibilidade ao contexto é a capacidade de uma aplicação ou *software* de detectar e responder às mudanças do ambiente [Maamar et al. 2006]. Um sistema sensível ao contexto é capaz de adaptar suas operações ao estado corrente sem intervenção humana, consequentemente melhorando a usabilidade e eficiência deste sistema [Baldauf et al. 2007].

Em grids pervasivos, o escalonamento é uma tarefa que pode ser beneficiada com a inclusão de sensibilidade ao contexto através da coleta de dados sobre os recursos do grid e tomada de decisões baseadas nestes dados.

2.2. Apache Hadoop e Escalonamento

O *framework* Apache Hadoop é organizado numa arquitetura de mestre-escravo e possui dois serviços principais, o serviço de armazenamento (HDFS) e o de processamento (YARN). Cada serviço possui mestre e escravos independentes como apresentado na Figura 1, onde é possível notar que os serviços NameNode e ResourceManager, mestres do HDFS e YARN respectivamente, e os seus respectivos escravos DataNode e NodeManager. Outro componente que aparece na figura é o ApplicationMaster, este componente é responsável pelo gerenciamento interno de cada *job*, também chamado de escalonamento de *tasks* ou tarefas. Entende-se *task* como uma fração do processamento das aplicações, ou seja, cada tarefa de Map ou Reduce corresponde a uma *task*. Enquanto o ApplicationMaster gerencia as tarefas, é função do ResourceManager gerenciar os *jobs* ou aplicações. Finalmente, o último componente a aparecer na figura é o Container, o qual representa uma alocação de recursos em um nó qualquer do *cluster*. A importância do container vem do fato de que todas as tarefas são executadas em uma instância de container.

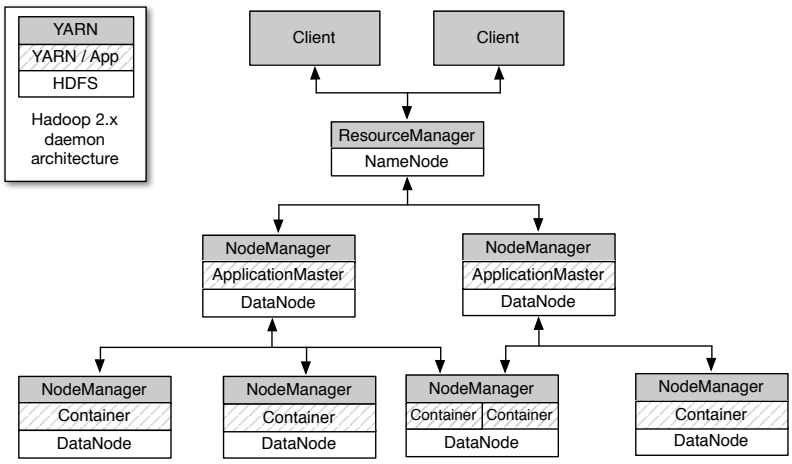

Figura 1. Arquitetura geral do Apache Hadoop

Apesar de existirem dois tipos de escalonamento no Hadoop, o escalonamento com mais opções de alteração é o de aplicações. O Hadoop disponibiliza alguns escalonadores de aplicação, os quais serão referidos apenas por escalonadores a partir de agora.

O escalonador mais simples é o Hadoop Internal Scheduler, é um escalonador que utiliza o algoritmo FIFO e tem boa performance em *clusters* onde não existe competição por recursos. Outro escalonador disponível é o Fair Scheduler, utilizado principalmente para o processamento de lotes de aplicações pequenas e rápidas. Este escalonador utiliza um escalonamento de dois níveis, buscando a divisão justa dos recursos [Hadoop 2013b]. A terceira opção, e também o padrão do Hadoop nas últimas versões, é o Capacity Scheduler. O Capacity Scheduler foi projetado para a utilização compartilhada do Hadoop e busca a maximização do *throughput* e da utilização do *cluster*. Seu funcionamento baseia-se em garantias mínimas de capacidade para os usuários, deslocando as capaci-

dades dos usuários inativos para aqueles que estão utilizando o *cluster*. Esta estratégia fornece elasticidade com um bom custo benefício [Hadoop 2013a].

A existência destes escalonadores adiciona flexibilidade no gerenciamento do *framework*. Apesar disso, os escalonadores disponíveis não detectam nem reagem à dinamicidade e heterogeneidade do ambiente, um requisito presente em ambientes pervasivos.

2.3. ZooKeeper

O ZooKeeper é um projeto da Apache e possui compatibilidade com o Hadoop. Inicialmente, o ZooKeeper foi implementado como um componente do Hadoop e virou um projeto próprio conforme cresciam suas funcionalidades e sua utilização em outras aplicações. Ainda, o ZooKeeper fornece ferramentas eficientes, confiáveis e tolerantes à falha para a coordenação de sistemas distribuídos [Hunt et al. 2010].

No caso deste trabalho, utiliza-se os serviços do ZooKeeper para monitorar e transmitir as informações de contexto coletadas nos nós escravos. No ZooKeeper existe um componente chamado *znode* onde é possível inserir qualquer tipo de dado e todos as instâncias de clientes e servidores (no caso de um servidor replicado) possuem acesso à este *znode*. A comunicação é feita através de processos que atualizam e processos que lêem o conteúdo destes componentes.

2.4. Trabalhos Relacionados

Ao longo dos anos diversos trabalhos propuseram melhorias para os mecanismos de escalonamento do Hadoop visando uma melhor performance de acordo com suas necessidades. Estas contribuições, na maior parte, podem ser divididas entre dois tipos: propostas de novos métodos de escalonamento ou propostas de melhoria na distribuição de recursos.

Os trabalhos [Kumar et al. 2012, Tian et al. 2009, Rasooli and Down 2012] assumem que a maior parte das aplicações executadas num cluser é periódica e possui cargas de CPU, memória, disco e rede similares. Estas suposições permitem que as aplicações e os nós sejam analizados e classificados de acordo com suas necessidades e suas capacidades nestas características. Uma vez classificados, o escalonamento torna-se um problema de combinar nós e applicações de mesma classificação. Seguindo nesta linha, o trabalho [Isard et al. 2009] propõe a utilização de um gráfo de capacidade-demanda que auxilia o cálculo do escalonamento ótimo com base em uma função de custo.

Embora os trabalhos apresentados até aqui focam na melhora da perfromance através da utilização de informação estática sobre recursos e aplicações, existem trabalhos que buscaram incoporar informações sobre as tarefas nas suas propostas. Os trabalhos [Zaharia et al. 2008, Chen et al. 2010] tentaram melhorar a distribuição das tarefas de uma aplicação buscando reduzir seu tempo de resposta em grandes *clusters*. Os autores em [Zaharia et al. 2008] utilizam heuristicas para inferir o progresso estimado de tarefas e fazer decisões sobre o lançamento de tarefas especulativas. Tarefas especulativas são cópias de tarefas já inicializadas quando existe suspeita de falha na tarefa original ou simplesmente a existência de lentidão no processamento. Já o trabalho [Chen et al. 2010], propõe a utilização de dados históricos de execução para melhorar a tomada de decisões.

O resultado final da utilização dos dois métodos – novos métodos de escalonamento e melhoria na distribuição de recursos – é um rebalanceamento de carga, forçando

nós mais rápidos a processarem mais dados e diminuindo a carga de trabalho em nós mais lentos. O trabalho [Sandholm and Lai 2010] busca o rebalanceamento de carga através de um sistema baseado na lei de oferta e demanda, permitindo que cada usuário influencie diretamente o escalonamento por meio de suas taxas de gasto. O principal objetivo é para permitir um compartilhamento de recursos dinâmico e baseado em preferências que os próprios usuários configuram.

Há também o trabalho [Xie et al. 2010], que tenta fornecer uma melhora na performance em aplicações através da melhora na colocação dos dados, utilizando principalmente a localização de dados como informação para a tomada de decisão. O ganho de performance é alcançado através, também, do rebalanceamento de carga nos nós. Esta proposta reduz o número de tarefas especulativas e transferência de dados pela rede.

3. Desenvolvimento

Através de um estudo aprofundado do escalonamento do Apache Hadoop, identificou-se uma estratégia para melhoria no processo sem a inserção de métodos intrusivos ou grande modificação nas políticas de escalonamento já implementadas no *framework*. A implementação realizada pode ser separada em duas tarefas distintas, coleta de dados e transmissão de dados.

3.1. Coletor de Contexto

O Apache Hadoop utiliza arquivos XML como método de configuração do *cluster*, cada nó possui alguns arquivos de configuração e cada arquivo possui diversas propriedades que podem ser alteradas. As informações referentes aos recursos disponíveis em dado nó também estão dentro deste conjunto de propriedades, forçando o administrador a configurar um arquivo para cada nó do *cluster*. Ainda, estas informações são transmitidas ao escalonador somente na inicialização do serviço, não ocorrendo qualquer tipo de atualização até que o serviço seja reinicializado. Estas limitações são gravíssimas num ambiente pervasivo, o qual sofre alterações durante a execução de uma aplicação, e portanto, necessita de um mecanismo que atualize as informações durante a execução.

Para solucionar este problema optou-se pela integração de um módulo de coleta de dados no Hadoop, o qual permite a coleta das informações sobre os recursos em um dado momento. O coletor foi desenvolvido para o projeto PER-MARE [STIC-AmSud 2014] e é baseado na API padrão de monitoramento do Java [Oracle 2014] permitindo coletar facilmente as informações de um nó sem a adição de bibliotecas externas. Esta API permite que informações como o número de processadores (cores) e a memória do sistema sejam identificadas, através da utilização de um conjunto de interfaces e classes abstratas que generalizam o processo de coleta. Devido à sua estrutura, pode-se facilmente integrar novos coletores para outras informações caso haja necessidade, como por exemplo a utilização de disco ou CPU.

3.2. Comunicação Entre Processos

Para que as informações adquiridas através do módulo de coleta possam ser utilizadas é necessário que estas cheguem até o processo do escalonador, em execução na máquina que possui o serviço Resource Manager. A escolha para a implementação desta comunicação no sentido escravo-mestre foi feita visando a compatibilidade com o Hadoop e a não

intrusão nos processos de comunicação já definidos. Para isso optou-se por utilizar uma estrutura de HashTable distribuída (DHT) no *znode*, na qual os escravos atualizam e o mestre pode ler as informações atualizadas.

O funcionamento se dá da seguinte forma: todos os escravos possuem uma *thread* chamada NodeStatusUpdater, esta *thread* coleta dados sobre a disponibilidade de recursos do nó a cada 30 segundos (periodo parametrizável) e, se a quantidade de recursos disponíveis estiver diferente da última leitura, a DHT do *znode* será atualizada. Concorrente a isto, o mestre possui uma *thread watcher* que observa a DHT e caso esta seja atualizada, a *thread watcher* será notificada e atualizará as informações no escalonador de acordo com a nova informação lida na DHT.

Esta solução implementa a capacidade de observação e atualização da disponibilidade de recursos em tempo real, melhorando a capacidade de adaptação do *framework*.

4. Experimentos e Resultados Parciais

A seguir, encontram-se descritos os experimentos realizados. A descrição foi dividida em duas subseções, uma para explicação do ambiente de testes e outra com resultados e análise.

4.1. Preparação do Ambiente

Primeiramente, configurou-se o *framework* Hadoop no *cluster genepi* do Grid'5000 [Grid 5000 2013]. O ambiente de execução foi configurado com quatro escravos, cada um possuindo a seguinte configuração: 2 CPUs Intel(R) Xeon(R) E5420 2.50 GHz (totalizando 8 cores por nó) e 8 GB de memória RAM. Todos nós do experimento possuíam o sistema operacional Ubuntu-x64-12.04, com a JDK 1.8 instalada e a versão 2.6.0 do Hadoop configurada.

O *benchmark* foi feito com a aplicação TeraSort, aplicada a um conjunto de dados de 15GB. Os recursos considerados nos experimentos foram a memória e o número de cores, uma vez que estes são os parâmetros utilizados pelo Capacity Scheduler para a alocação de tarefas (*containers*). Foram tomadas precauções para que nenhum outro serviço ou aplicação influenciasse os testes. As informações sobre a execução dos *containers* foi extraída por meio de análise das logs do Hadoop.

Após a implementação das melhorias no *framework* os seguintes casos de teste foram criados e configurados para os experimentos:

Caso A: representa a situação ideal, na forma de um *cluster* Hadoop dedicado, onde o usuário possui acesso à todos os recursos do *cluster* em qualquer momento. Isto implica que os recursos informados ao escalonador **sempre** corresponderão aos recursos disponíveis para o Hadoop. Consideram-se recursos informados como os dados que o escalonador utiliza para realizar suas políticas de escalonamento, enquanto, recursos disponíveis são aqueles estão livres e/ou sendo utilizados pelo próprio Hadoop. Utilizando uma notação percentual, os recursos informados são de 100% e os recursos disponíveis são de 100% durante toda execução.

Caso B: representa a situação decorrente do compartilhamento dos nós do cluster com outros usuários. Como consequência do compartilhamento, é possível que em, algum momento, ocorra uma inconsistência entre a quantidade de recursos informada e

disponível. Este caso aplica o comportamento padrão do Hadoop, no qual os recursos são informados por meio de arquivos XML **somente** na inicialização do serviço e nunca são atualizados. Em notação percentual, os recursos informados são de 100%, porém os recursos disponíveis são de 50%. Para simular este caso, optou-se por reduzir o número de recursos disponíveis (através da exclusão de nós) sem alterar a informação passada ao escalonador no Caso A.

Caso C: repete as especificações do Caso B, porém possui a implementação da proposta na forma de coletores de contexto e comunicação entre nós. Este caso simula quando uma outra aplicação é lançada **antes** da ocorrência da coleta e transmissão de dados, ou seja, quando uma nova aplicação for submetida ao *cluster*, este já estará com os dados atualizados. Em notação percentual, os recursos informados são de 50% e os recursos disponíveis são de 50%.

Caso D: representa uma extensão do Caso C em que a inicialização de outra aplicação ocorre **após** a coleta e transmissão dos dados e **antes** da submissão de uma aplicação, ou seja, a aplicação será lançada numa situação onde o *cluster* possui a informação errada (Caso B) e terá de se adaptar à nova configuração dos recursos (Caso C) durante a execução. Em notação percentual, os recursos informados no início da aplicação são de 100%, enquanto os recursos disponíveis são de 50%. Após a coleta e transmissão de dados os recursos informados também passam a ser 50%.

4.2. Resultados e análise

Os resultados dos experimentos estão resumidos na Tabela 1 e na Figura 2. Na Tabela 1, as colunas representam, respectivamente, da esquerda para direita: os casos explicados na seção anterior, o tempo total utilizado por todas as tarefas de Map, o tempo médio de execução das tarefas de Map, o desvio padrão do tempo médio de cada caso e o número de tarefas especulativas lançadas.

Como mencionado anteriormente todas tarefas são processadas em *Containers*, porém alguns destes não são afetados pelo escalonamento como por exemplo o ApplicationMaster e as tarefas de Reduce. Por esta razão estas tarefas foram ignoradas e a análise foi feita como foco nas tarefas de Map, as quais são afetadas pelo escalonamento sensível ao contexto.

Tabela 1. Resumo dos resultados

Caso	Tempo total(s)	Tempo médio (s)	Desvio padrão (s)	Tarefas especulativas
A	149	39.47	15.73	2
B	788	222.97	59.86	1
C	348	38.38	18.09	3
D	477	68.42	29.91	1

Foram gerados quatro diagramas de Gantt, um para cada caso. Nestes diagramas, cada linha representa os recursos de um nó do cluster consolidados por escala de cor. Quanto mais escuro o tom maior a carga de processamento, sendo o tom preto uma indicação de dezesseis containers em execução enquanto o tom branco representa zero containers. Ainda, cada linha é segmentada para indicar o término ou início de um *container* naquele instante. Os diagramas estão com escala em segundos e todos vão de zero

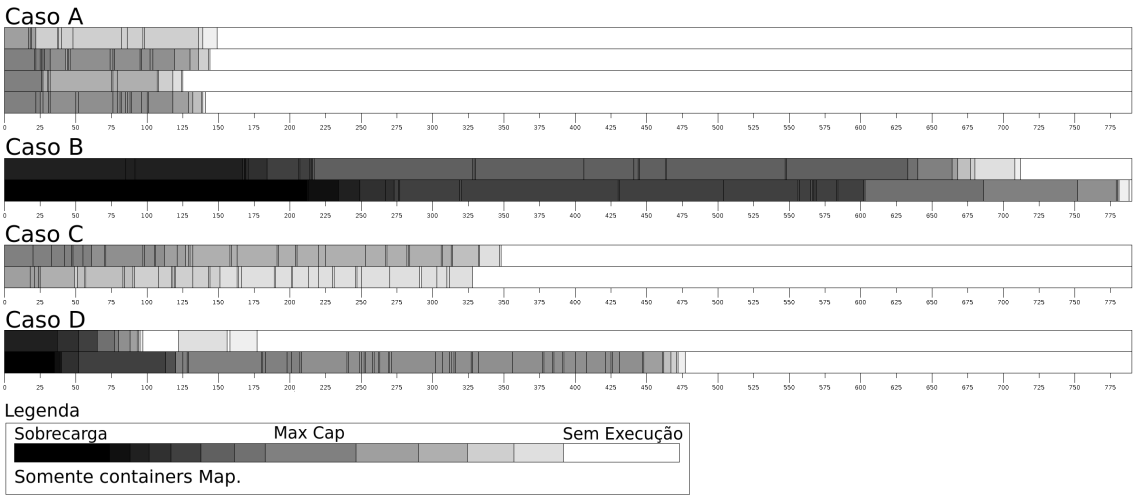

Figura 2. Diagramas de Gantt dos experimentos

a setecentos e oitenta segundos. Como mencionado na descrição dos casos, os casos B, C e D executam com apenas metade dos nós do caso A para simular a redução dos recursos.

Pela análise da Tabela 1, é possível notar que os casos A e C, onde os recursos reais são conhecidos antes do início das aplicações, possuem os menores tempo médio de execução e desvio padrão. Isto deve-se ao fato que estes nós nunca foram sobrecarregados, já que o escalonador possuia a informação correta. É possível notar este mesmo comportamento nos diagramas, onde os casos A e C possuem tons similares. Além disso, o caso C apresenta o comportamento esperado, uma vez que possuía metade dos recursos de A e demorou o dobro do tempo. Ainda na Tabela 1 também é possível notar que a quantidade de tarefas especulativas lançadas estavam entre 1 e 3 em todos casos, apesar de inicialmente parecer uma surpresa nos casos B e D é um comportamento esperado, uma vez que a decisão de lançar ou não uma tarefa especulativa é baseada numa comparação com as outras tarefas em execução, e nestes casos todas tarefas estavam lentas.

Nota-se também que os casos B e D possuem um tom escuro no início, significando que dezesseis containers (o dobro da capacidade real) estão sendo executados simultaneamente. Ainda, os primeiros containers nos casos A e C levaram em média 20 segundos para executar enquanto no caso B foram necessários 70 segundos, evidenciando uma sobrecarga nos nós. Embora tanto B quanto D possuísem as mesmas condições inicialmente (recursos disponíveis de 50% e recursos informados de 100%), o caso D demorou menos tempo para completar. Este melhora foi possível devido à coleta dos recursos atualizados e informação ao escalonador, permitindo que este reorganizasse as tarefas após o primeiro conjunto terminasse para que não houvesse sobrecarga. É possível confirmar esta afirmação ao comparar os tons do caso D, escuros somente no início, com os do caso B, escuros durante toda execução devido à falta de informação atualizada. Embora o escalonador não faça preempção de tarefas, é possível notar uma melhora de performance de cerca de 40% baseada unicamente no fato do escalonador evitar a sobrecarga dos nós.

Os casos C e D mostram que atualizações de contexto regulares contribuem para a redução do tempo de execução num cluster dinâmico que utiliza o Hadoop. Provou-se que, mesmo iniciando a execução no pior caso possível, a atualização de informações

auxilia o escalonador a minimizar o tempo de execução. A solução proposta neste trabalho contribui tanto com o fornecimento de informação correta antes do início da execução quanto com a adaptação da execução às variações de recursos.

5. Próximas atividades

O planejamento das próximas atividades inclui a execução de novos testes com características de carga de trabalho diferentes (CPU-Bound, IO-Bound, etc.) nos quatro casos já apresentados, além da inclusão de dois novos casos referentes à agregação de recursos no decorrer da execução. O intuito de novos testes, é para confirmar se a solução adotada apresenta uma melhora no desempenho para tipos de aplicação variados.

Para estes novos testes, optou-se por utilizar a ferramenta HiBench que é um pacote de *benchmarks* para *clusters* Hadoop, incluindo tanto *micro-benchmarks* quanto aplicações reais [Huang et al. 2010]. O Hibench possui *benchmarks* das categorias: *web search* (Page Rank e Nutch Indexing), aprendizagem de máquina (classificação bayesiana e K-means), pesquisa analítica (Hive Join e Hive Aggregation). Esta diversidade de aplicações apresenta testes com perfis de utilização de recursos variados, fornecendo um instrumento de teste adequado para a confirmação da utilidade das melhorias implementadas.

Referências

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277.

Chen, Q., Zhang, D., Guo, M., Deng, Q., and Guo, S. (2010). Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, pages 2736–2743, Washington, DC, USA. IEEE Computer Society.

Dean, J. and Ghemawat, S. (2008). Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113.

Grid 5000 (2013). Grid 5000. `https://www.grid5000.fr/`. Ultimo acesso: Agosto 2015.

Hadoop (2013a). Capacity scheduler. `http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html`. Ultimo acesso: Agosto 2015.

Hadoop (2013b). Fair scheduler. `http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html`. Ultimo acesso: Agosto 2015.

Huang, S., Huang, J., Dai, J., Xie, T., and Huang, B. (2010). The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 41–51.

Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the USENIX Annual Technical Conference*, pages 11–11, Boston, MA. USENIX Association.

Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. (2009). Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM*

SIGOPS 22nd symposium on Operating systems principles, SOSP '09, pages 261–276, New York, NY, USA. ACM.

Kumar, K. A., Konishetty, V. K., Voruganti, K., and Rao, G. V. P. (2012). Cash: context aware scheduler for hadoop. In *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, ICACCI '12, pages 52–61, New York, NY, USA.

Maamar, Z., Benslimane, D., and Narendra, N. C. (2006). What can context do for web services? *Commun. ACM*, 49(12):98–103.

Nascimento, A. P., Boeres, C., and Rebello, V. E. F. (2008). Dynamic self-scheduling for parallel applications with task dependencies. In *Proceedings of the 6th International Workshop on MGC*, MGC '08, pages 1:1–1:6, New York, NY, USA.

Oracle (2014). Overview of java se monitoring and management. `http://docs.oracle.com/javase/7/docs/technotes/guides/management/overview.html`, . Ultimo acesso: Agosto 2015.

Parashar, M. and Pierson, J.-M. (2010). Pervasive grids: Challenges and opportunities. In Li, K., Hsu, C., Yang, L., Dongarra, J., and Zima, H., editors, *Handbook of Research on Scalable Computing Technologies*, pages 14–30. IGI Global.

Rasooli, A. and Down, D. G. (2012). Coshh: A classification and optimization based scheduler for heterogeneous hadoop systems. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, SCC '12, pages 1284–1291, Washington, DC, USA. IEEE Computer Society.

Sandholm, T. and Lai, K. (2010). Dynamic proportional share scheduling in hadoop. In *Proceedings of the 15th International Conference on Job Scheduling Strategies for Parallel Processing*, JSSPP'10, pages 110–131, Berlin, Heidelberg.

Steffenel, L. A., Flauzac, O., Charão, A. S., Barcelos, P. P., Stein, B., Nesmachnow, S., Pinheiro, M. K., and Diaz, D. (2013). Per-mare: Adaptive deployment of mapreduce over pervasive grids. In *Proceedings of the 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC '13, pages 17–24, Washington, DC, USA. IEEE Computer Society.

STIC-AmSud (2014). PER-MARE project. `http://cosy.univ-reims.fr/PER-MARE`. Ultimo acesso: Agosto 2015.

Su, X. and Swart, G. (2012). Oracle in-database hadoop: When mapreduce meets rdbms. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 779–790, New York, NY, USA. ACM.

Tian, C., Zhou, H., He, Y., and Zha, L. (2009). A dynamic mapreduce scheduler for heterogeneous workloads. In *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, GCC '09, pages 218–224, Washington, DC, USA. IEEE Computer Society.

Xie, J., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., Yin, S., and Qin, X. (2010). Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In *Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*.

Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., and Stoica, I. (2008). Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA. USENIX Association.