Computação intensiva em dados com MapReduce em ambientes oportunistas

Jonhnny Weslley Silva[1], **Thiago Emmanuel Pereira**[1],
Carla de Araújo Souza[1], **Francisco Brasileiro**[1]

[1]Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Laboratório de Sistemas Distribuídos
Av. Aprígio Veloso, s/n - Bodocongó
Campina Grande - PB - 58.429-900, Brazil

{jonhnny,thiagoepdc,carla}@lsd.ufcg.edu.br, fubica@dsc.ufcg.edu.br

Abstract. *The MapReduce programming model has emerged as a popular approach for the processing and generation of large data sets. Considering the opportunistic scenario, the execution of such data intensive application can harm user experience and impact application's makespan as a consequence of naive data placement and scheduling. To deal with these problems, we have proposed and analysed data allocation strategies which consider resource volatility. We also have modified and evaluated BashReduce, a light weight implementation of MapReduce model based on UNIX bash tools. Our modified version had its performance evaluated against Hadoop showing a* 36% *speedup on application's makespan.*

Resumo. *O modelo de programação MapReduce tornou-se popular no processamento de aplicações intensivas em dados. Considerando a utilização de recursos oportunistas, a execução deste tipo de aplicação pode perturbar a experiência do usuário, bem como ter seu desempenho prejudicado caso o escalonamento de dados e computação não seja adequado. Para tratar estes problemas, desenvolvemos e avaliamos heurísticas de alocação de dados que levam em consideração a volatilidade dos recursos. Além disto, modificamos e avaliamos BashReduce, uma implementação leve do modelo MapReduce. O desempenho desta versão modificada foi comparado com Hadoop resultando em uma redução de* 36% *no tempo de resposta das aplicações executadas.*

1. Introdução

O modelo de programação MapReduce foi proposto inicialmente pela Google Inc. como suporte à aplicações de busca na web [Dean and Ghemawat 2008]. Desde então, o modelo tem sido aplicado em um conjunto amplo de aplicações tais como: mineração de dados, aprendizado de máquina e processamento de linguagem natural [PoweredByHadoop 2010].

Atualmente, dois arcabouços para o desenvolvimento e execução de aplicações conforme o modelo MapReduce são bastante populares: o arcabouço proprietário desenvolvido pelo Google Inc. e o Hadoop [Hadoop 2010], uma implementação de

código aberto mantida pela *Apache Software Foundation*. A implementação Google do MapReduce utiliza como sistema de armazenamento o Google File System (GFS) [Ghemawat et al. 2003], enquanto Hadoop utiliza o Hadoop Distributed File System (HDFS) [HDFS 2010]. Ambas as implementações foram projetadas para usar *clusters* de computadores de uso geral.

Um cenário distinto consiste em utilizar recursos não-dedicados, por exemplo, *desktops* conectados em redes locais. Esta abordagem, que tem apelo econômico em instituições que não possuem recursos suficientes para manter infraestruturas dedicadas de processamento, tem sido aplicada tanto em projetos de computação científica [Anderson et al. 2002, Woltman et al. , Shirts and Pande 2000] como corporativos [Chien et al. 2003, Platform Computing 2007, Data Synapse 2007, Univa 2007]. Embora traga ganhos econômicos, a utilização de recursos não-dedicados pode acarretar em perda de desempenho: na medida em que estes recursos são voláteis há um incremento na probabilidade de perda de trabalho útil e consequentemente aumento do tempo de resposta das aplicações, isto agravado pelo fato destas aplicações terem tipicamente uma vida curta [Kondo et al. 2007].

Neste trabalho, consideramos quão adequada é a execução do modelo de programação MapReduce em ambientes de computação não dedicados. Como primeiro passo na solução desta questão, avaliou-se como uma alocação de dados ciente da volatilidade dos recursos pode melhorar o desempenho de aplicações que processam grandes quantidades de dados. Em seguida, avaliou-se uma implementação do MapReduce para sistemas operacionais *POSIX* baseada em *scripts bash*, o BashReduce [BashReduce 2009]. Esta implementação foi projetada para ser simples e impor poucos requisitos à infraestrutura necessária para a computação. Em particular, BashReduce não assume a existência de um sistema de arquivos descentralizado que armazene os dados de forma distribuída, tal como o HDFS ou GFS, o que reduz bastante o desempenho da solução. O BashReduce foi modificado para utilizar o BeeFS, um sistema de arquivos distribuído que utiliza o espaço livre das estações de trabalho de redes locais [da Cunha Silva 2010, Souza et al. 2010] para armazenar dados.

De maneira similar ao GFS e HDFS, o BeeFS foi projetado seguindo um modelo de distribuição híbrido. Esse modelo possui um servidor centralizado para o armazenamento de metadados e gerenciamento da política de alocação de arquivos, e múltiplos servidores de dados para armazenar colaborativamente os dados do sistema. Diferentemente do GFS e HDFS, o BeeFS é um sistema de arquivos aderente à especificação *POSIX*, deste modo sendo uma solução adequada para o BashReduce. É importante levar em consideração que os dados armazenados em sistemas que não são aderentes ao padrão *POSIX* não podem ser acessados sem que as aplicações sejam modificadas. Deste modo, caso outras aplicações acessem os dados a serem processados, estes precisam ser gravados também em outro sistema de arquivos. Este desperdício pode ser considerável para instituições que não dispõem de recursos suficientes.

O tempo de resposta das aplicações MapReduce executadas usando a versão original do BashReduce foi comparado com a versão modificada que utiliza o BeeFS. Os resultados da avaliação mostram que a versão modificada apresenta uma melhoria no tempo de resposta variando entre 17 e 42 vezes quando comparada à versão original. Além disto, esta versão modificada apresentou uma diminuição de 36% no tempo de resposta das

aplicações executadas em comparação com Hadoop. Esses resultados demonstram que o BashReduce pode ser uma alternativa eficaz ao Hadoop para aplicações de curta duração. A sobrecarga apresentada por Hadoop tende a apresentar menor impacto em aplicações de maior escala.

O resto deste artigo é organizado da seguinte forma. Na próxima seção, é mostrado o efeito de estratégias de alocação no tempo de resposta de aplicações intensivas em dados sobre ambientes oportunistas. Nas Seções 3 e 4, o funcionamento da ferramenta BashReduce e do BeeFS são detalhados. As modificações realizadas na ferramenta BashReduce para utilizar o BeeFS são descritas na Seção 5. Os resultados da avaliação de desempenho da versão modificada do BashReduce são reportados na Seção 6. Por fim, na Seção 7 são discutidas conclusões e perspectivas de trabalhos futuros.

2. Processamento paralelo de grandes quantidades de dados usando recursos não-dedicados

Neste trabalho consideramos um cenário em que aplicações paralelas são executadas em recursos não-dedicados formados por *desktops* em redes locais. Por este motivo, existe a preocupação quanto à intrusividade, ou seja, a experiência do usuário de uma máquina que faz parte desta rede pode ser prejudicada devido à execução não solicitada das aplicações submetidas.

Nesta seção, descrevemos heurísticas de alocação de dados que operam de modo a aumentar as chances de que os arquivos estejam disponíveis para processamento em estações de trabalho ociosas. Foram definidas 6 heurísticas, nomeadas: all, equalizer, maxAvail, eqMaxAvail, meanAvail e eqMeanAvail. As heurísticas desenvolvidas foram avaliadas via simulação. Os resultados destas simulações foram comparados em termos do tempo de execução das aplicações e da quantidade de espaço armazenado necessário.

2.1. Modelo do sistema de processamento paralelo

Consideramos que o sistema de processamento paralelo utiliza um sistema de armazenamento o qual mantém um conjunto de arquivos com tamanho variável segundo uma dada distribuição. Cada arquivo é armazenado de maneira redundante através de replicação. O número de cópias é definido pelo nível de replicação. Os arquivos podem ser acessados a partir de qualquer máquina. Isto implica que eles podem ser acessados de forma local ou remota. Os acessos locais são mais rápidos do que os acessos remotos, obedecendo a seguinte relação:

$$t_r(A) = t_l(A) \times k,$$

onde $t_r(A)$ é o tempo estimado para realizar o acesso remoto para um arquivo A, $t_l(A)$ é o tempo estimado para realizar o acesso local para o arquivo A e k é o fator de impacto para realizar acesso remoto aos arquivos, onde $k > 1$.

As aplicações executadas são compostas por um conjunto independente de tarefas e recebem como entrada um conjunto de arquivos. Cada arquivo é processado por uma tarefa. Além disso, apenas uma única tarefa pode ser executada por uma máquina qualquer em um dado momento.

A disponibilidade é o intervalo de tempo que uma máquina permanece ociosa. Consideramos que uma máquina está disponível se ela não estiver sendo utilizada por

algum usuário. A disponibilidade de uma máquina é definida por um par de valores que representam o momento em que a máquina entrou em ociosidade e a duração de tempo no qual ela permaneceu neste estado. Formalmente, temos:

$$D_M = \{< d_1, \Delta t_1 >, < d_2, \Delta t_2 >, \ldots, < d_n, \Delta t_n >\}$$

onde D_M é o conjunto de tuplas que definem todos os períodos de disponibilidade da máquina M. Cada tupla possui um par de valores, $< d_n$, $\Delta t_n >$, os quais correspondem ao instante de tempo no qual a máquina ficou ociosa e a duração do tempo de ociosidade da máquina, respectivamente. Uma máquina é considerada indisponível se ela não estiver ociosa. Uma tarefa será executada em uma determinada máquina apenas se esta encontrar-se disponível. Porém, uma máquina pode torna-se indisponível durante a execução de uma tarefa, ou seja, antes da tarefa encerrar a sua execução. Neste caso, a execução é abortada e uma nova execução desta tarefa será escalonada. Desta forma, garante-se que a execução de tarefas não interfira nas atividades do usuário.

2.2. Modelo de simulação

Nas simulações realizadas foram empregados rastros reais de disponibilidade de *desktops* conectados em uma rede local [Kondo et al. 2004]. Com base nestes rastros, foi possível simular um sistema constituído de 244 máquinas por um período de 14 dias seguindo o modelo de disponibilidade descrito anteriormente.

O sistema de armazenamento simulado é formado por um conjunto de 244 máquinas que atuam como servidores de dados. Este sistema possui um conjunto de 1000 arquivos cujos tamanhos variam conforme uma distribuição uniforme entre $500MiB$ e $2GiB$. Os arquivos possuem cópias armazenadas em um ou mais servidores de dados dependendo do nível de replicação. Adotou-se um valor constante de 4.87 como fator de impacto no acesso remoto. Este fator de impacto foi calculado empiricamente em uma rede local conectada por uma rede Ethernet de $100Mbps$.

Em aplicações de uso intensivo de dados, o tempo de execução da aplicação é tipicamente uma função do tamanho dos dados de entrada. Neste caso, foi adotado o mesmo fator de conversão utilizado em [Ranganathan and Foster 2002], definido por:

$$t(F) = 300 \times S(F)$$

onde $t(F)$ é o tempo de processamento do arquivo F e $S(F)$ representa o tamanho do arquivo F em GiB.

Para cada cenário, foi simulada a execução de 300 aplicações. A quantidade de tarefas por aplicação varia conforme uma distribuição uniforme entre 3 e 10. As tarefas de uma aplicação são mapeadas por um escalonador para as máquinas que realizarão o processamento. Neste trabalho será adotado o escalonador Storage Affinity [Santos-Neto et al. 2004], no qual as tarefas são escalonadas nas máquinas de acordo com a afinidade entre elas. Essa afinidade corresponde à quantidade dos dados de entrada da tarefa, em *bytes*, que já está armazenada na máquina onde o arquivo está localizado.

2.3. Heurísticas para alocação de arquivos

Neste trabalho, foram elaboradas 6 heurísticas para alocação de arquivos: all, $equalizer$, $maxAvail$, $eqMaxAvail$, $meanAvail$ e $eqMeanAvail$. Os detalhes do funcionamento de cada uma delas serão descritos a seguir.

A heurística all cria cópias de cada arquivo em todas as máquinas disponíveis. Esta heurística resulta no menor tempo de resposta possível para a execução das aplicações, uma vez que maximiza a probabilidade de que uma máquina que contenha os dados esteja ociosa. Entretanto, esta heurística não é viável devido aos custos de armazenamento associados, servido como referencial de comparação com as demais heurísticas.

A heurística $equalizer$ cria réplicas dos arquivos de forma a balancear o espaço de armazenamento nas máquinas que participam do sistema.

A heurística $maxAvail$ cria cópias dos arquivos nas máquinas mais disponíveis do sistema, em outra palavras, aquelas que acumulam os maiores períodos de ociosidade.

A $eqMaxAvail$, assim como a anterior, cria cópias dos arquivos nas máquinas mais disponíveis do sistema. Porém, ela também realiza um balanceamento de réplicas visando reduzir a criação de $hotspots$. Uma vez que algumas máquinas possuem níveis de disponibilidade significativamente maiores do que outras, estas máquinas podem rapidamente concentrar um grande número de réplicas.

A heurística $meanAvail$ cria réplicas dos arquivos nas máquinas que possuem as maiores médias da duração dos períodos de disponibilidade. Com isto, evita-se a formação de $hotspots$ em máquinas com grandes intervalos de disponibilidade que não se repetem com muita frequência. De maneira semelhante à $eqMaxAvail$, a heurística $eqMeanAvail$ promove o balanceamento de réplicas com respeito à heurística $meanAvail$.

2.4. Comparação das heurísticas de alocação quanto ao tempo de resposta e demanda de armazenamento

Foram consideradas duas métricas de desempenho: o tempo de resposta das aplicações e a quantidade de dados que precisam ser armazenados. O tempo de resposta da aplicação é definido como o intervalo de tempo compreendido entre a submissão da aplicação e o momento em que a última tarefa escalonada finalizou sua execução. A segunda métrica é importante para entender os compromissos na redução do tempo de resposta e o consumo de recursos disponíveis. Uma vez que é possível reduzir o tempo de resposta de uma aplicação por meio do aumento do nível de replicação dos arquivos. Neste trabalho, foram simulados cenários considerando níveis de replicação que variavam de 1 até 244, valor máximo para o nível de replicação considerando o número de máquinas do sistema simulado.

Para que os resultados das simulações ficassem estatisticamente válidos, foi preciso simular pouco mais de 1000 cenários distintos. As médias dos resultados obtidos têm um erro máximo de 5% para mais ou para menos, com um nível de confiança de 95%. A Figura 1 mostra o gráfico *box plot* dos tempos de resposta das aplicações considerando cada uma das heurísticas de alocação de arquivos apresentadas neste trabalho. Por sua vez, a Figura 2 mostra a comparação dos tempos médios de resposta das aplicações em função do nível de replicação para cada uma das heurísticas.

De acordo com os resultados mostrados, o tempo de resposta das aplicações varia conforme o nível de replicação dos arquivos, como esperado, exceto para a heurística all que, para cada arquivo, cria cópias em todas as máquinas ignorando o nível de replicação. Por isso, esta heurística apresenta tempos de resposta praticamente constantes, indepen-

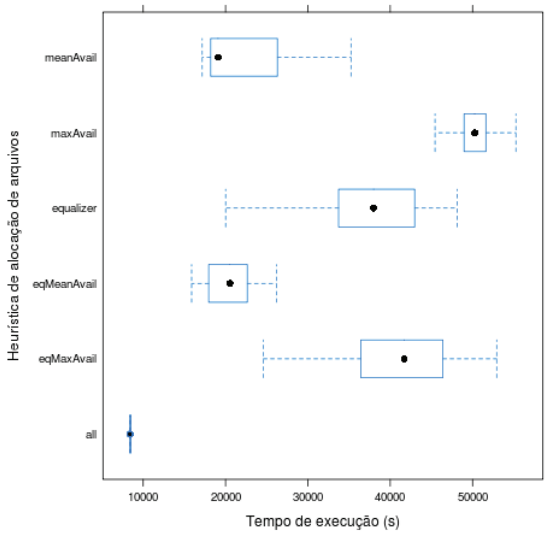

Figura 1. Gráfico box plot das médias do tempo de resposta das aplicações considerando cada uma das heurísticas para alocação de arquivos.

dentemente do nível de replicação utilizado. Porém, este desempenho custa caro, devido à grande quantidade de espaço de armazenamento necessária para armazenar as cópias dos arquivos em todas as máquinas. A heurística de alocação $equalizer$, não possui bons resultados para pequenos níveis de replicação. Contudo, ela apresenta ganhos significativos de desempenho quando o nível de replicação possui valores acima de 20% do número total de máquinas. Comportamento similar pode ser percebido na heurística $eqMaxAvail$. Já as heurísticas $meanAvail$ e $eqMeanAvail$ apresentam ganhos de desempenho relevantes sem necessitar de altos valores de replicação, logo, elas são mais indicadas para cenários onde espaço de armazenamento não é abundante. Entretanto, vale salientar que, ao contrário da heurística $eqMeanAvail$, a heurística $meanAvail$ não reduz o tempo de resposta das aplicações à medida que o nível de replicação aumenta. Por fim, a heurística $maxAvail$ apresenta um péssimo desempenho quando comparada às demais heurísticas. Este fenômeno é facilmente explicado pelo fato de que as máquinas mais disponíveis armazenam uma quantidade maior de cópias dos arquivos. Deste modo, uma vez que poucas máquinas são responsáveis por manter uma grande quantidade de cópias, são criados $hotspots$ durante a execução das aplicações.

3. Visão geral da ferramenta BashReduce

BashReduce é uma implementação feita em *bash script* do modelo de programação MapReduce projetada para utilizar estações de trabalho conectadas em uma rede local. Ele simplifica a execução de aplicações MapReduce ao permitir que as aplicações paralelas sejam escritas usando as ferramentas do *shell Unix* (tais como: `sort`, `awk`, `grep`, `cut`, `paste` entre outras). BashReduce utiliza apenas dependências aderentes ao padrão *POSIX* [Tanenbaum and Woodhull 2006], com isto reduz as barreiras de adoção relacionadas ao aprendizado de um conjunto novo de tecnologias.

Além disso, ainda que uma infraestrutura dedicada MapReduce esteja disponível, se os dados a serem processados não foram gerados nesse sistema, talvez seja mais van-

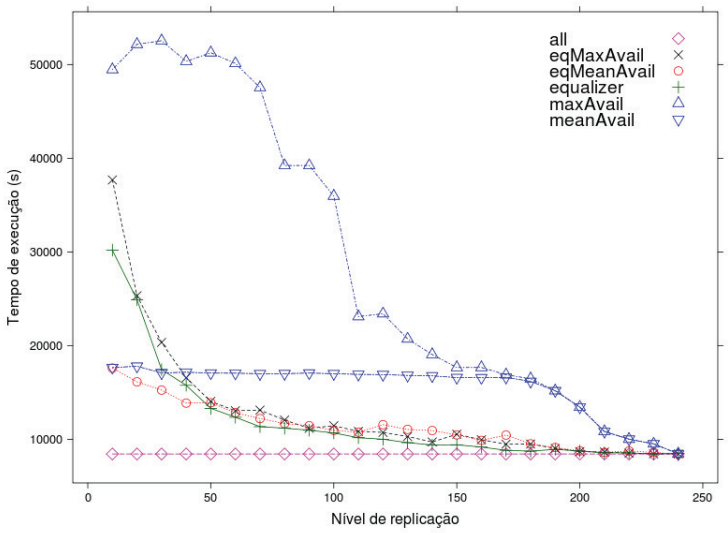

Figura 2. Comparação das heurísticas de alocação de arquivos em relação ao tempo de resposta das aplicações em função do nível de replicação.

tajoso utilizar o aplicativo BashReduce para processar estes dados sem a necessidade de transferi-los para a infraestrutura. Esta abordagem é particularmente interessante para aplicações que não são executadas mais de uma vez para o mesmo conjunto de dados.

BashReduce transparentemente gerencia o particionamento dos dados, paraleliza a computação, coleta os resultados produzidos, juntando-os para produzir o resultado final. Contudo, o aplicativo BashReduce não possui suporte para tolerância a falhas, logo não é indicado para executar aplicações que necessitam de várias horas para terminar o processamento dos dados. Mesmo assim, BashReduce ainda é uma ferramenta que pode ajudar em tarefas diárias, como por exemplo, realizar análise dos *logs* de outras aplicações [OpenDNS 2009].

Com BashReduce, o usuário pode utilizar como entrada um único arquivo ou um diretório — neste último caso, todos os arquivos localizados no diretório serão processados — para produzir um único arquivo com o resultado da computação realizada. Uma aplicação BashReduce é expressa com alguns poucos argumentos, dos quais, os mais importantes são o arquivo de entrada, o arquivo de saída, a lista de máquinas que serão utilizadas para realizar o processamento e os programas *map* e *reduce*. O Código 3 mostra a linha de comando de uma simples aplicação BashReduce:

```
$> br -i arquivo_entrada.txt -o arquivo_saida.txt \
    -h``host1 host2 host3 host4'' \
    -m ``grep string_de_busca'' -r ``uniq -c''
```

Exemplo de uma aplicação BashReduce simples.

Na aplicação BashReduce mostrada acima, o BashReduce irá dividir o arquivo `input.txt` e enviar os pedaços para as máquinas especificadas pela opção "-h" (`host1`, `host2`, `host3` e `host4`). Em cada uma destas máquinas o programa *map* (`grep pattern`) vai processar o fragmento do arquivo de entrada enviado para a respectiva máquina. A saída produzida pelos programas *map* servem de entrada para o pro-

grama *reduce* (`uniq -c`), cujos resultados produzidos serão armazenados no arquivo `output.txt` na máquina que disparou o comando BashReduce.

Mais detalhadamente, quando o usuário executa um comando BashReduce, ocorre a seguinte sequência de ações:

1. O aplicativo BashReduce usa o comando *netcat* (`nc`) para preparar os canais de comunicação com cada máquina especificada pelo usuário (*escravos*); durante a computação, cada máquina *escrava* deve possuir duas conexões abertas; enquanto uma delas recebe dados para serem processados, a outra conexão envia os dados produzidos para a máquina onde o comando BashReduce foi executado (*mestre*);
2. Os programas *map* e *reduce* são disparados remotamente nas máquinas *escrava* através do comando *secure shell* (`ssh`); e ficam aguardando pelos dados de entrada enviados pela máquina *mestre* para iniciar o processamento;
3. Os arquivos de entrada são particionados em H fragmentos, onde H é o número de máquinas *escravas* especificadas pelo usuário; após o particionamento, o aplicativo BashReduce inicia a transferência dos fragmentos para cada uma das máquinas *escravas*;
4. À medida que as máquinas *escravas* completam o processamento dos programas *map* e *reduce*, elas enviam os dados produzidos para a máquina *mestre*, onde o aplicativo BashReduce concatena todos os resultados produzidos para produzir um único arquivo — por padrão, a concatenação é realizada pelo programa *sort*, mas o usuário pode especificar outro programa para realizar a concatenação.

O BashReduce ainda possui um segundo modo de utilização no qual os fragmentos que compõem a entrada são armazenados em um sistema de arquivos distribuído. Dado que a hierarquia de diretório é acessível pelos *escravos*, estes podem acessar os arquivos sem a necessidade de transferência de dados com a máquina que disparou o comando BashReduce. Para tanto, o *mestre* necessita repassar para os *escravos* os nomes dos arquivos a serem processados. Embora o gargalo causado pela transferência de dados seja removido da máquina que dispara o comando, um novo é criado caso o sistema de arquivos distribuído empregado utilize um servidor centralizado – tal como é comumente utilizado por implantações típicas do sistema.

4. BeeFS

O BeeFS possui três componentes principais: um servidor centralizado de metadados (*queenbee*), servidores de dados (*honeycombs*) e clientes (*honeybees*). Estes componentes seguem um modelo de distribuição híbrido que mistura aspectos de sistemas cliente-servidor e entre-pares, o que permite i) reduzir a carga de trabalho no componente centralizado e ii) atender o aumento da demanda de forma granular, através da adição de novos servidores de dados [Souza et al. 2010].

O servidor de metadados é um componente confiável que deve ser instalado, preferencialmente, em uma máquina dedicada. Além do armazenamento de metadados, também é responsável por mapear a localização dos arquivos armazenados, controlar o acesso aos arquivos e coordenar o mecanismo de replicação de dados.

Os servidores de dados armazenam colaborativamente as cópias dos arquivos servidos pelo BeeFS. Ao contrário do servidor de metadados, estes não residem em máquinas

dedicadas, mas nas estações de trabalho que fazem parte de uma rede local. Os servidores de dados são acessados pelos clientes, para que leituras e modificações nos arquivos sejam realizadas, ou por outros servidores de dados, quando as réplicas mantidas por estes precisam ter seus conteúdos atualizados.

Os processos dos usuários têm acesso aos arquivos por meio de uma interface *POSIX* implementada pelo cliente BeeFS (*honeybee*). Geralmente, clientes e servidores de dados coexistem em uma mesma máquina. Assim, o algoritmo de alocação de arquivos padrão do BeeFS explora essa possibilidade para fins de melhoria de desempenho. Arquivos são, sempre que possível, criados no servidor de dados que executa na mesma máquina onde o cliente que solicitou a criação está executando.

4.1. Tolerância a falhas

Considerando a natureza não-dedicada dos recursos em que clientes e servidores de dados estão distribuídos, fica evidente a necessidade de mecanismos que mantenham a consistência e a disponibilidade do sistema perante a ocorrência de falhas. No BeeFS, há mecanismos para tolerar falhas tanto nos servidores de dados quanto no servidor de metadados [Soares et al. 2009]. Os clientes não precisam de mecanismos de tolerância a falhas, já que eles não armazenam nenhum tipo de informação.

O servidor de metadados é, também, responsável pela detecção de falhas dos servidores de dados. Quando uma falha é detectada em um dado servidor, todos os grupos de replicação que foram afetados – os quais tinham uma réplica armazenada no servidor em questão – precisam ser reorganizados de modo que o nível de replicação dos arquivo seja mantido. Isto é feito através da substituição do servidor comprometido por outros servidores disponíveis no momento.

Por razões de desempenho, o servidor de metadados adota uma política semelhante ao GFS [Ghemawat et al. 2003], mantendo os metadados armazenados na memória e persistindo periodicamente as atualizações para o disco por motivos de tolerância a falhas. Existe a possibilidade de ocorrer dois tipos de falhas no servidor de metadados: transientes e permanentes. São consideradas falhas transientes aquelas que não comprometem a integridade do sistema, mas tornam o serviço indisponível. Neste caso, serviços de monitoramento podem ser utilizados para automaticamente reiniciar o servidor de metadados. Em caso de falhas permanentes, o BeeFS pode ser iniciado no modo de recuperação [Soares et al. 2009]. No modo de recuperação, ao invés de carregar as informações dos metadados a partir do disco, o servidor de metadados recebe, como parâmetro de entrada, a lista de endereços dos servidores de dados que constituíam o sistema. Então, inicia o processo de recuperação comunicando-se com cada um dos servidores de dados na lista.

5. BashReduce com BeeFS

Como descrito anteriormente, BashReduce pode ser utilizado de duas formas distintas. Na primeira, os dados de entrada são divididos e enviados a partir da máquina mestre para as máquinas escravas utilizando a rede local. Na segunda, apenas os nomes dos arquivos de entrada são enviados. Para tanto, essa última técnica necessita que as máquinas tenham acesso aos arquivos por meio de um sistema de arquivos que possua uma hierarquia global de nomes, tal como o NFS ou o BeeFS.

Empregar um sistema de arquivos que utilize múltiplos servidores pode melhorar significativamente o desempenho das aplicações. Por exemplo, é possível escalonar tarefas de modo que essas acessem os dados de maneira local. Para aproveitar esta vantagem da distribuição de dados no BeeFS, o programa BashReduce precisa descobrir a localização das máquinas onde os arquivos estão armazenados durante a execução de uma aplicação. Para tanto, o BashReduce foi modificado para que fizesse uso do atributo estendido *beefs.replicationGroup*. Atributos estendidos são um tipo especial de metadado usado pelo BeeFS que permitem associar informações arbitrárias na forma chave/valor aos arquivos [Santos-Neto et al. 2008].

Esse atributo contém, entre outras informações, o nome da máquina que armazena a réplica primária do arquivo em questão. A partir da localização das máquinas, o programa BashReduce poderá executar a aplicação diretamente nas máquinas que armazenam os arquivos de entrada, evitando a transferência de dados pela rede local. Além disso, se todos os arquivos de entrada estiverem armazenados no sistema de arquivos BeeFS, torna-se desnecessário que o usuário informe a lista de máquinas a serem utilizadas para executar a aplicação, já que o programa BashReduce descobrirá o nome das máquinas em tempo de execução.

6. Avaliação de desempenho do BashReduce

Nesta seção, será apresentado o resultado da avaliação de desempenho do BashReduce em contraste com a implementação Hadoop. Na primeira parte dessa avaliação, comparou-se a versão tradicional do BashReduce, que utiliza o NFS, com a versão modificada que faz uso do BeeFS. Em uma segunda avaliação, comparou-se o desempenho do BashReduce modificado com o Hadoop em dois modos de utilização, Hadoop Tradicional e Hadoop Streaming. O primeiro modo, mais comum, é baseado na escrita de aplicações paralelas utilizando a API Java disponibilizada pelo Hadoop, enquanto no segundo modo é possível utilizar executáveis, sendo mais semelhante ao modo de execução do BashReduce.

6.1. Comparação do desempenho do BashReduce usando NFS e BeeFS

Para verificar o ganho de desempenho potencial do BashReduce usando o BeeFS, foi realizado um experimento comparando os tempos de execução de uma aplicação BashReduce considerando dois sistemas de arquivos distribuído para armazenar os arquivos de entrada, o BeeFS e o NFS. O sistema NFS foi escolhido por representar o estado-da-prática para sistemas de arquivos distribuídos em uma LAN.

A aplicação utilizada no experimento é equivalente ao "Grep distribuído", descrito no artigo original que descreve o modelo de programação MapReduce [Dean and Ghemawat 2008]. *Logs* reais gerados por simulações realizadas em pesquisas anteriores foram utilizados como arquivos de entrada. Durante a fase *map* da aplicação, ocorrências de determinadas entradas são buscadas no arquivo de entrada, em seguida, durante a fase *reduce* o número de entradas localizadas na fase anterior é contabilizado. Adicionalmente, para gerar o resultado final, a aplicação executa um programa que foi desenvolvido para concatenar os resultados produzidos pelas máquinas escravas e produz o sumário do número de ocorrências para cada evento da simulação.

Para avaliar a escalabilidade de cada um dos sistemas de arquivos, os experimentos foram executados em três cenários distintos. Em cada cenário, o número de máquinas escravas utilizadas é proporcional à quantidade de dados a serem processados. Os cenários

usaram 5, 10 e 20 máquinas escravas para processar $5GB$, $10GB$ e $20GB$ de dados brutos, respectivamente.

Os experimentos foram realizados usando estações de trabalho conectadas por uma rede Ethernet de $100Mbps$. Cada estação de trabalho possui um processador Intel Core 2 Duo $2.40GHz$ com $2GB$ de memória RAM e $100GB$ de espaço em disco, executando o sistema operacional Ubuntu 9.04 (kernel $2.6.28$). Além disso, tanto o servidor de metadados do BeeFS quanto o servidor NFS foram instalados em uma máquina com configuração superior, a qual possui um processador Quadcore - Intel(R) Xeon(R) CPU E5310 $1.60GHz$ com $4GB$ de memória RAM, executando o sistema operacional Debian 5 (kernel $2.6.26$) com arquitetura de 64-bits.

Os experimentos foram repetidos pouco mais de 50 vezes, garantindo um nível de confiança de 95% com base nas médias dos tempos de execução mensurados, e com uma taxa de erro inferior a 5%. A Figura 3 mostra a média dos tempos de execução do experimento em função do número de máquinas escravas tanto para o BeeFS quanto para o NFS.

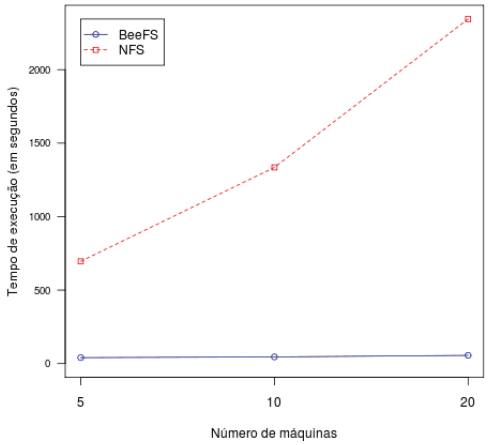

Figura 3. Tempo de execução (em segundos) para as aplicações BashReduce executando sobre o BeeFS e o NFS em função do número de máquinas escravas usadas.

Como esperado, à medida que o número de máquinas escravas é incrementado, o tempo de execução das aplicações BashReduce que utilizaram NFS também aumenta. Este resultado é explicado pela transferência dos dados pela rede a partir de um servidor centralizado, o qual rapidamente torna-se um gargalo. Assim, o atraso causado pela transferência dos dados representa a maior parte do tempo de execução das aplicações.

Por outro lado, as aplicações que executam utilizando BashReduce sobre o BeeFS apresentam praticamente o mesmo desempenho em todos os cenários avaliados, demonstrando a escabilidade linear do sistema. Esse grau de escalabilidade deve-se ao fato de que os dados são lidos diretamente dos discos locais das máquinas *slaves* em paralelo, reduzindo o tráfego da rede. Este resultado comprova a suposição de que uma arquitetura híbrida que utiliza servidores de dados distribuídos incrementa substancialmente o paralelismo ao passo que reduz o gargalo no servidor central.

6.2. Comparando BashReduce com Hadoop

Neste experimento, utilizou-se a mesma aplicação do experimento anterior e manteve-se a proporção de $1GB$ de dados a serem processados por cada máquina escrava. Foram utilizadas três configurações distintas: BashReduce sobre BeeFS, Hadoop Tradicional e Hadoop Streaming. Na configuração Hadoop Tradicional, as aplicações são submetidas usando a API Java disponibilizada pelo arcabouço. Na configuração Hadoop Streaming [HadoopStreaming 2010] são executados binários genéricos, por exemplo, utilitários *shell* do UNIX, semelhante ao que é feito com o BashReduce.

A instalação Hadoop foi configurada usando as opções padrão. As mais relevantes para esse experimento são: nível de replicação igual a 3, blocos de dados de $64MB$, e números máximos de funções *map* e *reduce* executando simultaneamente para cada *task tracker* igual a 8. Em todos os experimentos, o *namenode* do HDFS e o servidor de metadados do BeeFS foram instalados na mesma máquina.

O experimento foi conduzido em um *cluster* dedicado com 8 nós conectados por uma rede Ethernet $100Mbps$; cada nó possui dois processadores 4-core 2.67 GHz Intel Xeon rodando o sistema operacional 64-bits Ubuntu 9.04 com 8 GB de RAM e $500GB$ de espaço em disco. Um nível de confiança de 95% para a média do tempo de execução dos experimentos, com uma taxa de erro inferior a 5%, foi conseguido com 50 repetições do experimentos.

A Tabela 1 resume a eficiência relativa do BashReduce em comparação com o Hadoop. A sobrecarga apresentada pelo Hadoop tradicional, cerca de 56%, pode ser explicada pelas atividade de gerenciamento das tarefas em execução, por exemplo o monitoramento das máquinas escravas. Por sua vez, os resultados mostram que o Hadoop Streaming possui desempenho significativamente inferior, uma sobrecarga de 496.3% quando comparado com o BashReduce, isto devido à necessidade de comunicação entre o processo que constitui a aplicação e a JVM em que o Hadoop executa.

Tabela 1. Tempo de execução (em segundos) para cada configuração: BeeFS, Hadoop Tradicional e Hadoop Streaming

	Tempo de execução (s)	Sobrecarga em relação ao BeeFS (%)
BeeFS	61.7031	-
Hadoop Tradicional	96.2500	56.0
Hadoop Streaming	367.9500	496.3

7. Conclusão e Trabalhos Futuros

Neste trabalho consideramos os problemas relacionados com o uso do modelo de programação MapReduce em ambientes oportunistas. Nesta direção, mostramos que o uso de estratégias de alocação de dados cientes da volatilidade dos recursos pode reduzir o tempo de execução de aplicações paralelas que executam grandes quantidades de dados em ambientes oportunistas.

Também consideramos outra abordagem para este problema que consiste em fazer com que o sistema demande menos poder computacional, diminuindo assim a competição

por recursos com a aplicação executada, que, consequentemente, tem seu tempo de execução diminuído – além de reduzir as chances de perda de trabalho devido a indisponibilidade da máquina na qual executa. Seguindo esta abordagem, modificamos e avaliamos o BashReduce. Os experimentos realizados mostram que o tempo de resposta das aplicações MapReduce executadas usando a versão modificada apresentou uma melhoria entre 17 e 42 em comparação com a versão original. Além disto, esta versão modificada apresentou uma diminuição de 36% no tempo de resposta das aplicações executadas em comparação com Hadoop.

Consideramos que a inadequação das implementações atuais pode ser melhor entendida com uma avaliação de desempenho mais detalhada. Por exemplo, seria útil entender quanto cada fase do fluxo de execução contribui para o tempo de execução total. Esta avaliação pode apontar otimizações a serem feitas no código do Hadoop como alternativa à utilização do BashReduce. Ainda, uma alternativa promissora seria avaliar a utilização do Hadoop em conjunto com o BeeFS, ou outro sistema distribuído adequado para processamento paralelo que seja aderente ao padrão *POSIX*, com isto evitando os problemas decorrentes da adoção do HDFS em ambientes não dedicados.

Agradecimentos

Francisco Brasileiro é pesquisador do CNPq/Brasil.

Referências

Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61.

BashReduce (2009). Mapreduce bash script. `http://blog.last.fm/2009/04/06/mapreduce-bash-script`.

Chien, A., Calder, B., Elbert, S., and Bhatia, K. (2003). Entropia: architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63(5):597–610.

da Cunha Silva, T. E. P. (2010). Políticas de Alocação e Migração de Arquivos em Sistemas de Arquivos Distribuídos para Redes Locais. Master's thesis, Universidade Federal de Campina Grande, Campina Grande, Paraíba, Brasil.

Data Synapse (2007). Data synapse. `http://www.datasynapse.com/`.

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

Ghemawat, S., Gobioff, H., and Leung, S. T. (2003). The Google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43.

Hadoop (2010). Hadoop. `http://hadoop.apache.org`.

HadoopStreaming (2010). Hadoop streaming. `http://hadoop.apache.org/common/docs/current/streaming.html`.

HDFS (2010). Hadoop distributed file system (hdfs). `http://hadoop.apache.org/hdfs`.

Kondo, D., Chien, A., and Casanova, H. (2007). Scheduling task parallel applications for rapid turnaround on enterprise desktop grids. *Journal of Grid Computing*, 5(4):379–405.

Kondo, D., Taufer, M., III, C. L. B., Casanova, H., and Chien, A. A. (2004). Characterizing and evaluating desktop grids: An empirical study. *Parallel and Distributed Processing Symposium, International*, 1:26b.

OpenDNS (2009). "building opendns stats"at velocity. `http://rcrowley.org/2009/06/23/building-opendns-stats-at-velocity.html`.

Platform Computing (2007). Platform computing. `http://www.platform.com/`.

PoweredByHadoop (2010). Applications powered by hadoop. `http://wiki.apache.org/hadoop/PoweredBy`.

Ranganathan, K. and Foster, I. (2002). Decoupling computation and data scheduling in distributed data-intensive applications. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 352, Washington, DC, USA. IEEE Computer Society.

Santos-Neto, E., Al-Kiswany, S., Andrade, N., Gopalakrishnan, S., and Ripeanu, M. (2008). enabling cross-layer optimizations in storage systems with custom metadata. In *Proceedings of the 17th international symposium on High performance distributed computing*, pages 213–216. ACM.

Santos-Neto, E., Cirne, W., Brasileiro, F., and Lima, A. (2004). Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids. In *JSSPP*, pages 210–232.

Shirts, M. and Pande, V. (2000). Screen savers of the world unite! *Science*, 290(5498):1903.

Soares, A. S., Pereira, T. E., Silva, J. W., and Brasileiro, F. V. (2009). Um modelo de armazenamento de metadatados tolerante a falhas para o DDGfs (in portuguese) . In *WSCAD-SSC 2009: Proceedings of the 10th Computational Systems Symposium*.

Souza, C., Lacerda, A. C., Silva, J., Pereira, T., Soares, A., and Brasileiro, F. (2010). Beefs: Um sistema de arquivos distribuído posix barato e eficiente para redes locais. In *SBRC 2010 - Salão de Ferramentas*.

Tanenbaum, A. S. and Woodhull, A. S. (2006). *Operating Systems Design and Implementation*. Prentice Hall, 3rd edition.

Univa (2007). Univa. `http://www.univa.com/`.

Woltman, G. et al. GIMPS, The Great Internet Mersenne Prime Search.