Uma Avaliação do MapReduce para Inspeção Distribuída de Tráfego de Aplicações

Thiago Pereira de Brito Vieira, Paulo Fernando Almeida Soares, Stenio Flavio Lacerda Fernandes, Vinicius Cardoso Garcia

[1]Centro de Informática - Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851 - Cidade Universitária - 50.732-970 - Recife - PE

{tpbv, pfas, sflf, vcg}@cin.ufpe.br

Abstract. *Diagnosing distributed systems is a complex task. Network traffic analysis can be used to evaluate these systems, but it is needed mechanisms to deal with a growing amount of network traffic. MapReduce can be used for distributed traffic analysis, although network traces present characteristics not similar to the data type commonly processed through MapReduce. Due to lack of evaluation of MapReduce for traffic analysis, this paper evaluates the performance of MapReduce to packet level analysis and deep packet inspection, characterizing its scalability, speedup and the behavior of its phases. The experiments evidence the predominant phases in this kind of job, and show the impact of block, input and cluster size into job completion time and scalability.*

Resumo. *Diagnosticar sistemas distribuídos é uma tarefa complexa, a análise de tráfego de rede pode ser utilizada para avaliar estes sistemas, mas são necessários mecanismos capazes de avaliar uma crescente quantidade de tráfego de rede. MapReduce pode ser usado neste sentido, embora os traces de rede apresentem características diferentes dos tipos de dados geralmente processados por MapReduce. Devido às lacunas existentes quanto a avaliação do MapReduce para análise de tráfego de rede, este artigo avalia o desempenho do MapReduce para análise em nível de pacotes e inspeção profunda de pacotes (DPI), caracterizando sua escalabilidade, ganhos em tempo de execução e o comportamento de suas fases. Os experimentos executados evidenciam as fases predomimantes neste tipo de processamento e mostram o impacto causado pelo tamanho do cluster, do bloco e da entrada de dado no tempo de conclusão e escalabilidade.*

1. Introdução

Avaliar e diagnosticar sistemas distribuídos é uma tarefa complexa, trabalhos têm sido feitos para analisar a comunicação entre os componentes de um sistema distribuído, a fim de obter avaliações precisas, escaláveis e com baixo impacto [Sigelman et al. 2010, Sambasivan et al. 2011, Mi et al. 2012, Nagaraj et al. 2012]. É possível avaliar aplicações distribuídas através da análise do tráfego de rede, com pouco impacto no desempenho da aplicação avaliada e pouca dependência dos desenvolvedores, analisando o protocolo de comunicação, os fluxos, a taxa de transferência e a distribuição de carga entre os componentes de uma aplicação distribuída [Aguilera et al. 2003, Yu et al. 2011].

Para avaliar sistemas distribuídos a partir do tráfego de rede, é necessário fazer uma inspeção profunda de pacotes (Deep Packet Inspection - DPI) e obter informações

dos estados da aplicação, o que requer um esforço adicional em relação às abordagens tradicionais de DPI, que não costumam avaliar estados da aplicação. Embora trabalhos tenham sido feitos para aumentar o desempenho de DPI [Antonello et al. 2012], a avaliação de estados das aplicações reduz a capacidade de processamento de DPI para avaliar grande quantidade de tráfego de rede. Com o crescimento da velocidade dos enlaces de rede e da utilização de sistemas distribuídos para prover serviços na Internet [Sigelman et al. 2010], é necessário desenvolver abordagens capazes de lidar com um crescente volume de tráfego de rede e tornar eficiente a avaliação de aplicações através da análise de tráfego.

MapReduce [Dean and Ghemawat 2008] se tornou um importante modelo de programação e plataforma para processamento distribuído de grande massas de dados, com uso em diversas aplicações da indústria e academia [Guo et al. 2012]. MapReduce pode avaliar o tráfego de aplicações distribuídas, através da coleta de tráfego em pontos de um *datacenter* e o seu processamento *offline*, utilizando a análise em nível de pacotes [Lee et al. 2011], que avalia cada pacote individualmente para extrair informações da camada de rede ou transporte, e/ou DPI [Vieira et al. 2012b, Vieira et al. 2012a], que avalia cada bloco sem divisão para remontar dois ou mais pacotes e obter informações da camada de aplicação.

As principais avaliações sobre o desempenho do MapReduce focam no processamento de texto [Wang et al. 2009, Jiang et al. 2010, Chen et al. 2011], embora o MapReduce possa apresentar desempenho específico para cada tipo de dado processado [Tan et al. 2012, Groot 2012]. Devido às lacunas existentes sobre a avaliação do MapReduce para análise de tráfego de rede e às peculiaridades deste tipo de dado, que é binário e pode apresentar restrições quanto a sua divisão em blocos e em registros, é necessário entender o comportamento do MapReduce para processar tráfego de rede e quais otimizações podem ser feitas para explorar o potencial do MapReduce para avaliar sistemas distribuídos através da análise de tráfego. Este artigo avalia o MapReduce para análise em nível de pacotes e DPI, provendo as seguintes contribuições:

- Caracterização das fases do MapReduce para análise em nível de pacotes e DPI;
- Descrição da escalabilidade obtida com a adição de máquinas ao *cluster*;
- Identificação do melhor tamanho do bloco em relação ao tamanho do *cluster*;
- Avaliação dos ganhos obtidos ao utilizar MapReduce para DPI;
- Comparação entre duas abordagens do MapReduce para análise em nível de pacotes;

O restante deste artigo é organizado como o seguinte. A seção 2 descreve os trabalhos relacionados. A seção 3 apresenta a metodologia de avaliação e os resultados são descritos na seção 4. A seção 5 discute os resultados e a seção 6 conclui o artigo.

2. Trabalhos Relacionados

Algumas técnicas têm sido desenvolvidas para avaliar sistemas distribuídos através da análise de requisições, eventos e do comportamento de um sistema distribuído [Sambasivan et al. 2011, Mi et al. 2012, Nagaraj et al. 2012]. Aguilera *et al* (2003) propôs uma abordagem para isolar gargalos em sistemas distribuídos, baseado em registros das mensagens trocadas. Embora o tráfego de rede tenha sido utilizado para extrair as informações desejadas, esta abordagem não utiliza processamento distribuído. Yu *et*

al (2011) apresentou o SNAP, que avalia as interações entre as aplicações e a rede, coletando passivamente estatísticas do TCP e *logs* dos Sockets, provenientes de informações do sistema operacional. SNAP não extrai informações do tráfego de rede das aplicações, como também não utiliza computação distribuída. Nosso objetivo é utilizar computação distribuída para analisar tráfego de rede e extrair indicadores de aplicações distribuídas.

Lee *et al* (2011) apresentou uma ferramenta de processamento de *traces* de rede utilizando MapReduce. Nesta abordagem um bloco é dividido em pacotes e atribuídos individualmente às funções de *Map*, que são executadas em paralelo, sem informação compartilhada entre si. Na avaliação desta solução foi aplicada a análise em nível de pacotes, para extrair indicadores dos protocolos IP, TCP e UDP. Foi avaliado o tempo para a conclusão de *Jobs* sob diferentes tamanhos de dado de entrada e dois tamanhos de *cluster*, mas não foram avaliadas as fases do MapReduce, nem os impactos causados pela variação do tamanho do bloco. Esta abordagem não permite que mais de um pacote seja avaliado por função de *Map*, mas é possível iterar sob os dados gerados pelas funções de *Map* e *Reduce* e avaliar mensagens da camada de aplicação, embora isto aumente o tempo de execução dos *Jobs* e torne esta abordagem menos eficiente para DPI.

Em um trabalho anterior propusemos uma abordagem para a utilização do MapReduce para extrair indicadores de aplicações distribuídas a partir do tráfego de rede, utilizando DPI [Vieira et al. 2012b]. Para isto foi adotado o particionamento de um *trace* em arquivos com o tamanho definido pelo tamanho do bloco utilizado pelo HDFS (Hadoop Distributed File System), para cada bloco ser processado inteiramente e individualmente por uma função de *Map*. Foi feita uma avaliação sobre a efetividade e a capacidade de processamento do Mapreduce para DPI, mostramos que MapReduce pode expressar algoritmos de DPI e descrevemos a capacidade de processamento obtida com a configuração padrão do Hadoop, mas os principais fatores de configuração do MapReduce não foram analisados, como também não foi avaliado o comportamento de suas fases.

No melhor do nosso conhecimento, não encontramos trabalhos que avaliaram o desempenho do MapReduce com a variação dos seus principais fatores, para a análise de pacotes e DPI aplicados à avaliação de aplicações. Nosso trabalho mostra o comportamento das fases do MapReduce e avalia seu desempenho com a variação do tamanho do *cluster*, tamanho do bloco e tamanho do dado processado, para análise de pacotes e DPI.

3. Metodologia de Avaliação

Este artigo avalia a escalabilidade, ganhos em tempo de execução e as fases do MapReduce para análise em nível de pacotes e DPI. Foram capturados *traces* de aplicações distribuídas, que foram divididos em arquivos com tamanhos definidos pelo tamanho do bloco avaliado, para serem armazenados no HDFS, conforme descrito em [Vieira et al. 2012b].

Exemplos foram desenvolvidos para avaliar o MapReduce para análise de pacotes e DPI. *CountUpDriver* implementa análise de pacotes para um contador de portas, utilizando como entrada para as funções de *Map*, um arquivo com o tamanho definido pelo tamanho do bloco adotado pelo HDFS, sem a divisão do bloco em pacotes. Também foi avaliado um contador de porta implementado por *P3*, proposto por Lee *et al* (2011), com uma abordagem que divide um bloco em pacotes e processa cada um individualmente. *JxtaSocketPerfDriver* implementa DPI para extrair, de um *trace* de uma aplicação JXTA [Duigou 2003], o tempo de ida e volta (round-trip time - RTT) das mensagens JXTA, o

número de conexões por tempo e a taxa de transferência entre pares clientes e servidores. *JxtaSocketPerfDriver* processa um bloco inteiro como entrada para cada função de Map, para remontar as mensagens JXTA que tenham seu conteúdo dividido em muitos pacotes, devido a segmentação do TCP. Um pacote TCP pode transportar uma ou mais mensagens JXTA por vez, o que torna necessário avaliar toda a carga útil de um pacote para reconhecer todas as possíveis mensagens existentes, ao invés de avaliar apenas a assinatura de uma mensagem. O RTT das mensagens JXTA é calculado a partir do tempo entre o envio de uma mensagem e o recebimento da confirmação de recebimento desta mensagem, o que demanda a análise de muitos pacotes para avaliar estados da aplicação.

Para analisar o ganho com a adoção do MapReduce contra os mesmos algoritmos implementados por uma solução não-distribuída, foram desenvolvidos exemplos que processam o mesmo conjunto de dados e que implementam os mesmos algoritmos de *CountUpDriver* e *JxtaSocketPerfDriver*, mas que processam todos os dados em uma única máquina. Estes exemplos são respectivamente *CountUpMono* e *JxtaSocketPerfMono*. O código de todos exemplos desenvolvidos são livres e públicos[1].

Os *traces* analisados foram capturados usando Tcpdump e divididos em arquivos com tamanho do bloco de 32MB, 64MB e 128MB. Para avaliar o MapReduce para DPI de uma aplicação distribuída, foram capturados dois conjuntos de dados de tamanhos diferentes, a partir do tráfego de uma aplicação de transferência de arquivos implementada com JXTA-Socket, formada por seis máquinas. Foi coletado um conjunto de 30Gb de tráfego de rede, com dados divididos em 30 arquivos de 128MB, 60 arquivos de 64MB e 120 arquivos de 32MB, e outro conjunto de dados de 90Gb, dividido em 90 arquivos de 128MB, 180 arquivos de 64MB e 360 arquivos de 32MB. Durante a captura, cada cliente JXTA-Socket estabeleceu uma conexão com um servidor e enviou mensagens com conteúdos de tamanho entre 64KB e 256KB, cada servidor recebeu os dados enviados e confirmou o seu recebimento. Para avaliar MapReduce quanto a análise em nível de pacotes, foi capturado o tráfego da transferência entre clientes e um servidor de um serviço de armazenamento de dados em *Cloud Computing*. Neste caso foram adotados os mesmos tamanhos dos conjuntos de dados e tamanhos de arquivos usado para o caso de DPI.

Os experimentos adotaram os fatores e níveis descritos na Tabela 1, que representam a quantidade de máquinas alocadas, a quantidade de dados acumulados para a execução de um *Job* e o tamanho das tarefas a serem executadas. A partir dos resultados, foram avaliadas as métricas descritas na Tabela 2, que representam o impacto dos fatores e níveis, no desempenho e comportamento das fases do MapReduce.

Tabela 1. Fatores e Níveis

Fatores	Níveis
N. Máquinas	2 - 29
Tamanho do Bloco	32MB, 64MB e 128MB
Tamanho da Entrada	90Gb e 30Gb

Tabela 2. Métricas

Métricas	Descrição
Tempo de Conclusão	Tempo para a conclusão de um *Job*
Tempo das Fases	Tempo absoluto das fases no tempo de um *Job*
Distribuição das Fases	Tempo relativo das fases no tempo de um *Job*
Escalabilidade	Ganho de capacidade de processamento mediante a adição de máquinas ao *cluster*
Ganho	Ganho no tempo de conclusão em comparação à execução do mesmo algoritmo de forma não distribuída

Para avaliar as métricas selecionadas, foi desenvolvido o Hadoop-Analyzer [2], que extrai informações a partir dos *logs* do Hadoop e gera gráficos com os indicadores das

[1] http://github.com/tpbvieira/hadoop-dpi

[2] http://github.com/tpbvieira/hadoop-analyzer

métricas avaliadas, conforme apresentado na seção 4. Os experimentos foram executados em um *cluster* com 30 máquinas compostas de 4 núcleos de 3.2GHz, 8 GB de RAM e 260GB de espaço em disco, executando Linux com kernel 3.2.0-29. Foi usado o Hadoop-1.0.3 configurado para ter o máximo de 4 tarefas de *Map* e 1 tarefa de *Reduce* por máquina, também foi definido o parâmetro -Xmx1500m para a JVM e 400 para $io.sort.mb$. *CountUpDriver* e *JxtaSocketPerfDriver* tiveram o número de *Reducers* definido pelo número de *slots* para *Reducers* por máquina, calculado por $numReducers = (0.95)(numMaquinas)(maxReducersPorMaquina)$ [Kavulya et al. 2010]. O exemplo implementado com *P3* utilizou um número de *Reducers* fixado em 10 pela implementação disponibilizada por Lee *et al* (2011). Cada experimento foi executado 20 vezes para obter valores confiáveis [Chen et al. 2011], dentro de um intervalo de confiança de 95% e taxa de erro máxima de 5%.

4. Resultados

Cada conjunto de dado foi processado por *clusters* com uma variação no número de máquinas entre 2 e 29, com tamanho do bloco de 32MB, 64MB e 128MB. A Tabela 3 descreve o tempo de conclusão, em segundos, de *JxtaSocketPerfMono* e *CountUpMono* para processar conjuntos de dados de 90Gb e 30Gb, sob todos os fatores e níveis avaliados.

Tabela 3. Tempo de conclusão da execução não-distribuída para análise em nível de pacotes e DPI

	JxtaSocket(90Gb)	CountUp(90Gb)	JxtaSocket(30Gb)	CountUp(30Gb)
32MB	1.745,35	872,40	584,92	86,71
64MB	1.755,40	571,33	587,02	91,76
128MB	1.765,50	745,25	606,50	94,82

A Figura 1 mostra o tempo de conclusão do *JxtaSocketPerfDriver* e o ganho em tempo de conclusão contra o *JxtaSocketPerfMono* para DPI de 90Gb de um tráfego JXTA.

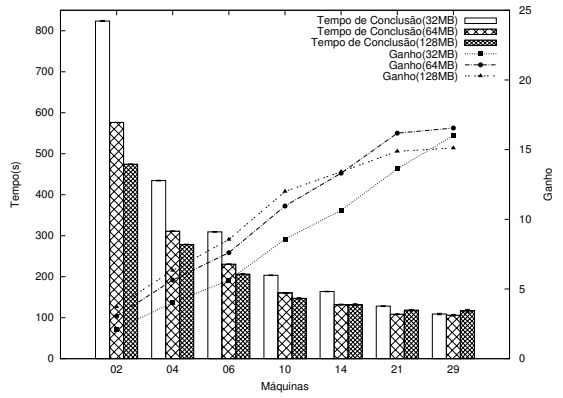


Figura 1. Tempo de conclusão e ganhos para extrair indicadores de 90Gb de tráfego JXTA por DPI

JxtaSocketPerfDriver apresentou melhor desempenho em relação a *JxtaSocketPerfMono* em todas variações dos fatores. Foi observado ganho de 3,7 vezes com 2 máquinas e bloco de 128MB, que foi incrementado com a adição de máquinas ao *cluster*, chegando a um ganho máximo de 16,19 vezes com 29 máquinas e bloco de 64MB. O ganho alcançado com bloco de 32MB foi o pior caso inicialmente, mas seu ganho aumentou com a adição de máquinas ao *cluster*, se tornando melhor que o obtido com blocos de 128MB e semelhante ao ganho obtido com blocos de 64MB, em *cluster* com 29 máquinas. A escalabilidade de processamento com blocos de 32MB mostrou redução no tempo

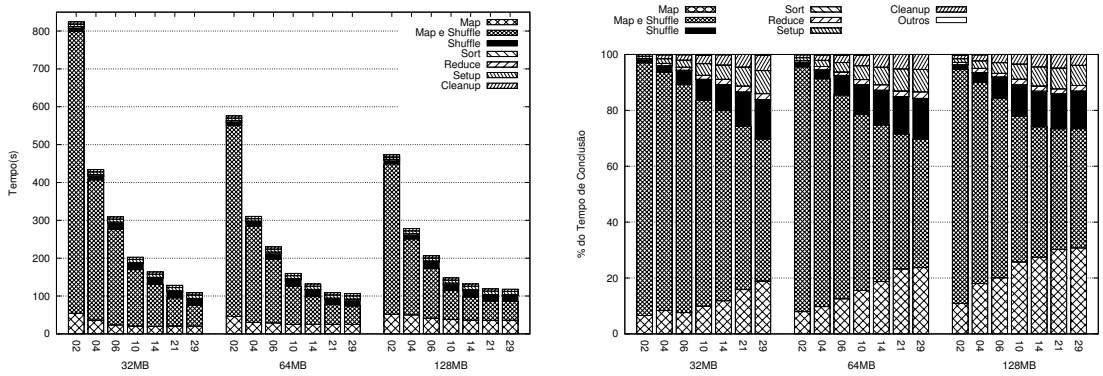
de conclusão em toda adição de máquina ao *cluster*, mas casos com blocos de 64MB e 128MB não apresentaram redução significativa no tempo de conclusão em casos com *clusters* maiores que 21 máquinas. De acordo com a Figura 1, o tempo de conclusão não reduziu linearmente com adição de máquinas, a redução no tempo de conclusão se tornou menos significativa em *clusters* com mais que 21 máquinas.

A capacidade de processamento para DPI de 90Gb de tráfego JXTA foi 159,89 Mb/s com 2 máquinas, aumentando para 869,43 Mb/s com 29 máquinas e bloco de 64MB, conforme exibido na Tabela 4. Embora a capacidade de processamento tenha aumentado com a adição de máquinas, a capacidade de processamento por máquina alocada reduziu.

Tabela 4. Capacidade de processamento com DPI de 90Gb de tráfego JXTA e blocos de 64MB

Máquinas	2	4	6	10	21	29
Segundos	576.38	310.93	230.57	160.5	108.38	106
Mb/s	159.89	296.40	399.70	574.20	850.34	869.43
(Mb/s)/máquina	79.94	74.10	66.61	57.42	40.49	29.98

A Figura 2 mostra o comportamento das fases do MapReduce para DPI de 90Gb, com variação no tamanho do bloco e *cluster*. A execução do MapReduce pode ser dividida em 4 fases: *Map*, *Shuffle*, *Sort* e *Reduce*. As tarefas de *Shuffle* podem ser iniciadas antes da conclusão de todas as tarefas de *Map*, permitindo a sobreposição entre as fases de *Map* e *Shuffle*, esta sobreposição é exibida como uma fase específica, a *Map e Shuffle*. O tempo consumido pela configuração (*Setup*) e limpeza (*Cleanup*) também foi representado, para uma melhor visualização da divisão do tempo de execução dos *Jobs*.



(a) Tempos absoluto ocupado pelas Fases

(b) Percentual do tempo ocupado pelas Fases

Figura 2. Comportamento das fases do MapReduce para DPI de 90Gb de tráfego JXTA

A Figura 2(a) mostra o tempo consumido por cada fase no tempo para conclusão de *Jobs* de DPI de tráfego JXTA. As tarefas de *Map* consumiram a maior parte do tempo, além de ter sido a fase mais afetada com a variação do número de máquinas, tendo seu tempo reduzido com a adição de máquinas ao *cluster*, embora esta redução tenha se tornado menos significativa em *cluster* com mais de 21 máquinas. O tempo de *Shuffle* apresentou baixa variação com adição de máquinas, como as fases de *Sort* e *Reduce*, que requerem baixo tempo de execução e não sofrem alteração significativa com a variação dos fatores. A Figura 2(b) exibe a porcentagem de cada fase no tempo de conclusão dos *Jobs*. Neste gráfico é considerada uma fase adicional, chamada *Outros*, a qual representa o tempo consumido para executar as demais tarefas de gerenciamento do *cluster*, como

o escalonamento e atribuição das tarefas. O comportamento seguido pela ocupação das fases foi similar nos três tamanhos do bloco analisados, com exceção do tempo de *Map e Shuffle* em *clusters* com mais de 21 máquinas e bloco de 128MB, que se manteve invariante com mudança no número de máquinas. Com o aumento do tamanho do *cluster*, foi observado uma redução da proporção de *Map* e um aumento das fases de *Shuffle*, *Setup* e *Cleanup*. Na Figura 2(a) foi observado que o tempo das fases de *Shuffle*, *Setup* e *Cleanup* consumiram um tempo sem alteração significativa com as variações dos fatores. Com a adição de máquinas e o decremento no tempo de conclusão, a proporção de tempo consumido por estas fases se tornou mais significativa em relação ao tempo total de execução, dado que o tempo de conclusão foi reduzido mas o tempo consumido por estas fases se manteve quase constante, passando a representar uma proporção maior no tempo de execução de *Jobs* com tempo de conclusão menor, conforme a Figura 2(b).

A Figura 3 mostra a comparação do tempo de conclusão entre *CountUpDriver* e *P3* para um contador de portas, através da análise em nível de pacotes de 90Gb de tráfego.

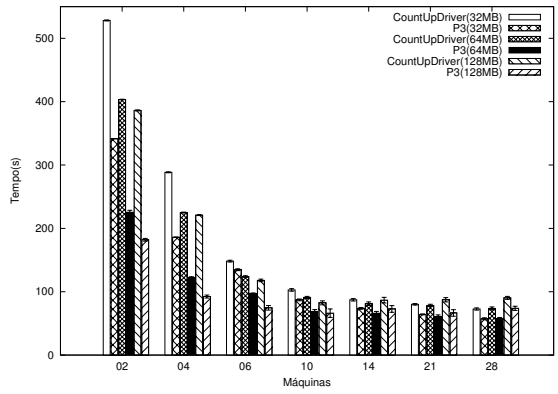


Figura 3. Comparação entre *CountUpDriver* e *P3* para análise em nível de pacotes de 90Gb

P3 obteve um melhor tempo de conclusão que *CountUpDriver* sob todos os fatores, mostrando que a divisão do blocos em pacotes proporcionou melhor desempenho para a análise em nível de pacotes. Além disto, pôde ser observado que o tamanho do bloco é um fator significativo para *CountUpDriver* e *P3*, por ter apresentado uma signficativa variação no tempo de conclusão em *clusters* menores. Com a variação do tamanho do *cluster*, foi observado que usando blocos de 128MB foi obtido um melhor tempo de conclusão em *clusters* com até 6 máquinas. Nos casos com blocos de 128MB a adição de máquinas ao *cluster* com mais de 10 máquinas, não implicou em ganhos significativos no tempo de conclusão. A utilização do blocos de 32MB ou 64MB apresentou desempenho significativamente diferente apenas até 14 máquinas, com ambos passando a apresentar um desempenho semelhante em *clusters* maiores, mas apresentando um desempenho melhor que o obtido com blocos de 128MB.

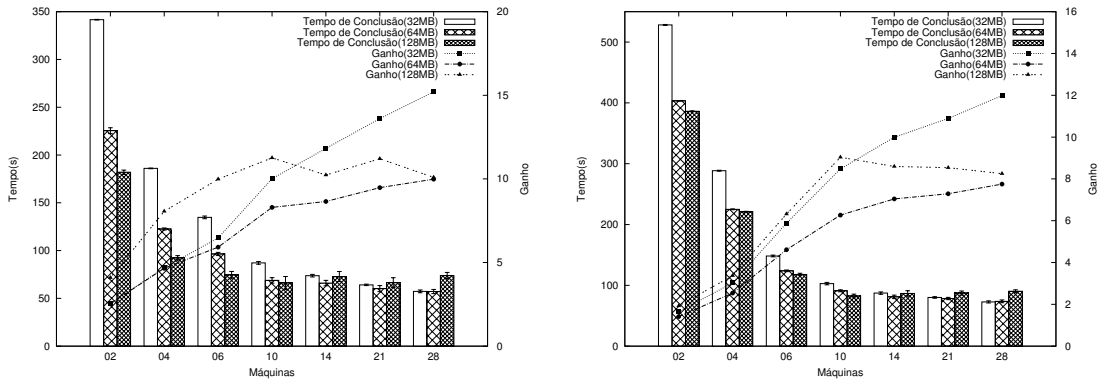
Tabela 5. Capacidade de processamento de *P3* para análise de pacotes de 90Gb e blocos de 64MB

Máquinas	2	4	6	10	21	28
Segundos	223.06	122.87	96.44	70.11	58.22	57.38
Mb/s	413.16	750.06	955.62	1314.50	1582.96	1606.13
(Mb/s)/máquina	206.58	187.51	159.27	131.45	75.37	55.38

A Tabela 5 exibe a capacidade de processamento de *P3* para analise em nível de pacote de 90Gb de tráfego de rede, utilizando blocos de 64MB. O pior caso obteve uma

capacidade de processamento de 413,16 Mb/s com 2 máquinas, enquanto o melhor caso obteve uma capacidade de processamento de 1.606,13 Mb/s utilizando 28 máquinas.

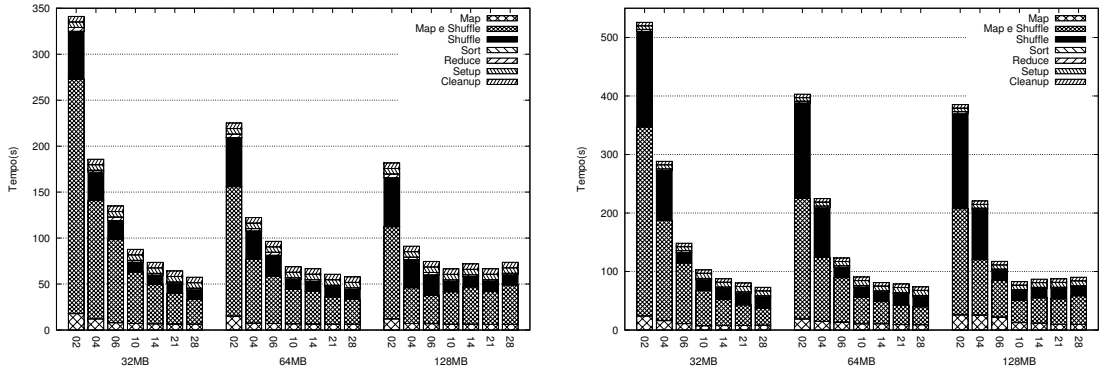
A Figura 4 exibe o tempo de conclusão e ganhos obtidos por *P3* e *CountUpDriver* comparados a *CountUpMono*, sob uma variação no tamanho do *cluster* e do bloco. Para ambos os casos, blocos de 128MB proporcionaram um melhor tempo de conclusão em *clusters* com até 6 máquinas, mas proporcionaram um tempo de conclusão pior em *clusters* com mais que 21 máquinas. A escalabilidade do ganho de tempo de conclusão foi melhor com blocos de 32MB, apresentando ganho em toda adição ao *cluster*, com melhores ganhos que os obtidos com blocos de 64MB ou 128MB em *clusters* com mais que 14 máquinas.



(a) P3 (b) CountUpDriver

Figura 4. Tempo de conclusão e ganhos de *P3* e *CountUpDriver* para análise de pacotes de 90Gb

As Figuras 5 e 6 mostram o comportamento das fases do MapReduce para a análise em nível de pacotes de um *trace* de 90Gb implementada por *P3* e *CountUpDriver*.



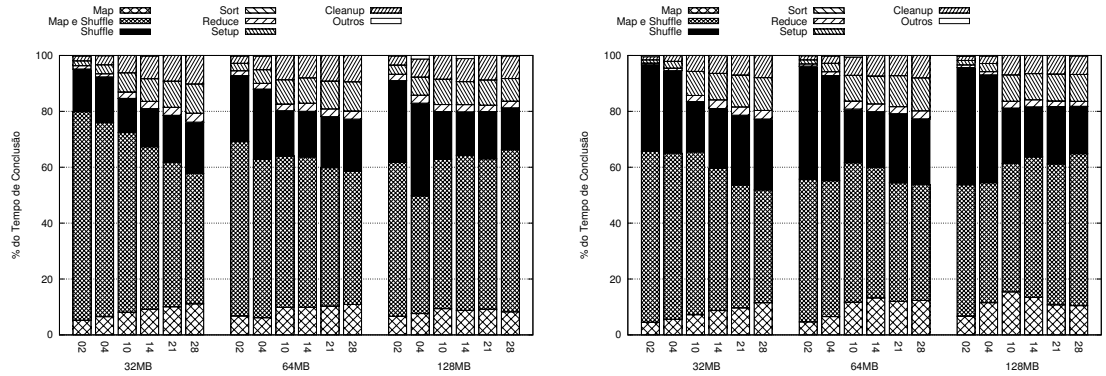
(a) P3 (b) CountUpDriver

Figura 5. Tempo absoluto das fases do MapReduce para análise em nível de pacotes de 90Gb

As Figuras 5(a) e 5(b) mostram o tempo consumido pelas fases durante a execução dos *Jobs*. O comportamento das fases é similar com o comportamento seguido pelas fases durante a análise de DPI, com a fase de *Map* sendo predominante, com o tempo das fases de *Map* e *Shuffle* não decrescendo com a adição de máquinas em *clusters* maiores que um tamanho específico, e com a fase de *Sort* e *Reduce* consumindo um tempo relativamente

baixo. A exceção é que a fase de *Shuffle* consome mais tempo na análise em nível de pacotes que em casos de DPI, principalmente em *clusters* menores. Isto acontece devido a quantidade de dados intermediários gerados serem maiores para a análise de pacotes, pois para cada pacote um valor intermediário é gerado. Já em casos de DPI, a geração de um dado intermediário pode ser proveniente do processamento de vários pacotes.

As Figuras 6(a) e 6(b) mostram o percentual das fases no tempo de conclusão de *P3* e *CountUpDriver* para análise de pacotes de 90Gb. Em ambos os casos as fases de *Map* e *Shuffle* consumiram uma porção maior que as outras fases, sob todos os fatores.



(a) P3 (b) CountUpDriver

Figura 6. Percentual de tempo das fases do MapReduce na análise em nível de pacotes de 90Gb

Para o conjunto de dados de 30Gb foram conduzidos os mesmos experimentos descritos para o conjunto de 90Gb. A avaliação das fases apresentou similaridade com os resultados dos experimentos para DPI e análise em nível de pacotes de 90Gb, mas diferenças significativas foram identificadas nos ganhos no tempo de conclusão e na escalabilidade, conforme exibido na Figura 7. O ganho máximo obtido para DPI de 30Gb foi de 7,90 vezes, com bloco de 32MB e 29 máquinas, enquanto foi obtido um ganho de 16,19 vezes no processamento de 90Gb. Na Figura 7 é possível observar que a capacidade de processamento apresentou escalabilidade significativa apenas até 10 máquinas, *clusters* maiores não apresentaram ganhos quando adotado bloco de 128MB. Ao adotar bloco de 64Mb ou 32MB, a adição de máquinas ao *cluster* apresentou ganhos pouco significativos.

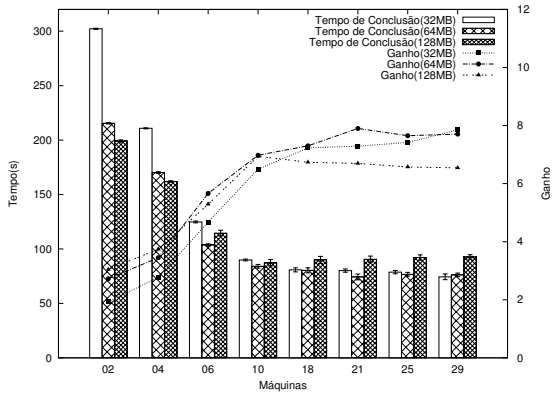


Figura 7. Tempo de conclusão e ganhos para extrair indicadores de 30Gb de tráfego JXTA por DPI

5. Discussão

O tamanho do bloco e do *cluster* foram importantes para a obtenção de melhor tempo de conclusão dos *Jobs* e para a escalabilidade da capacidade de processamento mediante a adição de máquinas. Avaliamos o impacto do tamanho do bloco para análise de pacotes e DPI, verificamos que blocos de 128MB proporcionaram um melhor tempo de conclusão em *clusters* menores, mas blocos de 64MB proporcionaram melhor desempenho em *clusters* maiores. Com o impacto do tamanho do bloco e do *cluster* no tempo de conclusão, é necessário avaliar a melhor alocação de máquinas em relação ao tamanho do bloco adotado, para obter eficiência no tempo de conclusão dos *Jobs* e alocação de recursos.

As diferentes capacidade de processamento obtidas para avaliar 30Gb e 90Gb evidenciaram que MapReduce apresentou melhor eficiência para o processamento de maiores quantidades de dados. Diante disto, pode ser mais eficiente acumular dados de entrada para processar uma grande quantidade de dados, a fim de melhor explorar o potencial de máquinas já alocadas e evitar desperdício de recursos. Este comportamento da capacidade de processamento também pode fornecer informações para quantificar a alocação de máquinas do *cluster* para determinados *Jobs*, em função to tamanho da entrada de dados.

A avaliação do tamanho do dado de entrada e do *cluster* é importante para alocar recursos eficientemente. Utilizando escalonadores como o *Hadoop Fair Scheduler* [Zaharia et al. 2010], é possível evitar desperdício de recursos através do agrupamento de máquinas de um *cluster* [Verma et al. 2012], para executarem determinados *Jobs*, permitindo quantificar o tamanho do grupo de acordo os tipos de *Jobs* que serão executados.

A comparação do tempo de conclusão entre *CountUpDriver* e *P3* mostrou que *P3*, que divide o *trace* em pacotes, apresentou melhor desempenho que *CountUpDriver*, que processa um bloco sem divisão. Ao processar um bloco inteiro, o *Mapper* tem a capacidade máxima de paralelismo definida pela quantidade de *slots* alocados por máquina, enquanto que a divisão do bloco em pacotes permite que cada pacote seja processado concorrentemente, por threads independentes, incrementando a concorrência por *Mapper*.

O número de ondas de execução influencia o tempo de conclusão de *Jobs*, pois a redução deste tempo está relacionada coma redução no número de ondas para a conclusão de todas as tarefas [Kavulya et al. 2010]. O número de ondas é definido pela quantidade de *slots* disponíveis para a execução das tarefas e a quantidade de tarefas necessárias; por exemplo, se um *Jobs* é dividido em 10 tarefas em um custer com 5 *slots*, serão necessárias 2 ondas de execução para concluir todas as tarefas. A Figura 7 exibe um caso de DPI de 30Gb, usando blocos de 128MB, em que não houve redução no tempo de execução devido a manutenção do número de ondas de execução. Mas outros casos em que houveram redução do número de ondas de execução apresentaram uma insignificante redução no tempo de conclusão, mediante adição de máquinas ao *cluster*, como os casos que usaram blocos de 128MB em *clusters* com mais de 21 máquinas para processar 90Gb.

A avaliação das fases do MapReduce para análise em nível de pacotes e DPI mostrou que a fase de *Map* foi prodominente, e que *Shuffle* foi a segunda fase mais representativa. A sobreposição entre *Map* e *Shuffle* representou mais que 50% do tempo dos *Jobs*. Um longo tempo da fase de *Map* implica em um longo tempo em que tarefas de *Shuffle* estarão sendo executadas até a conclusão da fase de *Map*, como ocorre na configuração padrão do Hadoop, em que a fase de *Shuffle* inicia após 5% da fase de Map ser concluída.

Casos com pouca geração de dados intermediários durante a fase de *Map* podem causar subutilização das tarefas de *Shuffle*, que poderiam ter seu início postergado e ter seus slots utilizados por outras tarefas e antecipar a conclusão do *Jobs*.

6. Conclusão

Neste artigo avaliamos o desempenho do MapReduce para a análise em nível de pacotes e DPI. Foram avaliados como o tamanho do dado utilizado como entrada, o tamanho do bloco e o tamanho do *cluster* geram impacto nas fases do MapReduce, no tempo de conclusão, na escalabilidade da capacidade de processamento e nos ganhos de tempo de conclusão, em relação ao mesmo algoritmo executado por uma única máquina.

A análise das fases do MapReduce mostrou que a análise em nível de pacote e DPI são atividades intensivas na fase de *Map*, que corresponde a mais de 70% do tempo total de execução. A fase de *Shuffle* é a segunda mais predominante, que ao se sobrepor a fase de *Map*, passa a representar um tempo significativo no tempo de conclusão dos *Jobs*.

Foi mostrado que o tamanho do dado de entrada, do bloco e do *cluster* são importantes fatores para obter um tempo de conclusão otimizado e para explorar eficientemente a capacidade de processamento e escalabilidade proporcionada pelo MapReduce, devido as variações no tempo de conclusão causadas pelas variações no tamanho do bloco e por não ocorrer, em alguns casos, aumento da capacidade de processamento com a adição de máquinas ao *cluster*. Também foi relatado que a abordagem que processa um bloco inteiro, sem divisão, proporcionou um desempenho inferior ao obtido com a abordagem que divide o bloco em pacotes e os processa individualmente em funções de *Map*.

Referências

Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P., and Muthitacharoen, A. (2003). Performance debugging for distributed systems of black boxes. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03.

Antonello, R., Fernandes, S., Kamienski, C., Sadok, D., Kelner, J., Godorc, I., Szaboc, G., and Westholm, T. (2012). Deep packet inspection tools and techniques in commodity platforms: Challenges and trends. *Journal of Network and Computer Applications*.

Chen, Y., Ganapathi, A., Griffith, R., and Katz, R. (2011). The case for evaluating mapreduce performance using workload suites. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 19th International Symposium on*.

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

Duigou, M. (2003). Jxta v2.0 protocols specification. Technical report, IETF Draft.

Groot, S. (2012). Modeling i/o interference in data intensive map-reduce applications. In *Applications and the Internet (SAINT), IEEE/IPSJ 12th International Symposium on*.

Guo, Z., Fox, G., and Zhou, M. (2012). Investigation of data locality and fairness in mapreduce. In *Proceedings of third international workshop on MapReduce and its Applications Date*, MapReduce '12.

Jiang, D., Ooi, B. C., Shi, L., and Wu, S. (2010). The performance of mapreduce: an in-depth study. *Proc. VLDB Endow.*

Kavulya, S., Tan, J., Gandhi, R., and Narasimhan, P. (2010). An analysis of traces from a production mapreduce cluster. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*.

Lee, Y., Kang, W., and Lee, Y. (2011). A hadoop-based packet trace processing tool. In *Proceedings of the Third international conference on Traffic monitoring and analysis*, TMA'11.

Mi, H., Wang, H., Yin, G., Cai, H., Zhou, Q., and Sun, T. (2012). Performance problems diagnosis in cloud computing systems by mining request trace logs. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*.

Nagaraj, K., Killian, C., and Neville, J. (2012). Structured comparative analysis of systems logs to diagnose performance problems. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12.

Sambasivan, R. R., Zheng, A. X., De Rosa, M., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., and Ganger, G. R. (2011). Diagnosing performance changes by comparing request flows. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11.

Sigelman, B. H., Barroso, L. A., Burrows, M., Stephenson, P., Plakal, M., Beaver, D., Jaspan, S., and Shanbhag, C. (2010). Dapper, a large-scale distributed systems tracing infrastructure. Technical report, Google, Inc.

Tan, J., Meng, X., and Zhang, L. (2012). Coupling scheduler for mapreduce/hadoop. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*, HPDC '12.

Verma, A., Cherkasova, L., Kumar, V., and Campbell, R. (2012). Deadline-based workload management for mapreduce environments: Pieces of the performance puzzle. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*.

Vieira, T., Soares, P., Machado, M., Assad, R., and Garcia, V. (2012a). Evaluating performance of distributed systems with mapreduce and network traffic analysis. In *ICSEA 2012, The Seventh International Conference on Software Engineering Advances*.

Vieira, T., Soares, P., Machado, M., Assad, R., and Garcia, V. (2012b). Measuring distributed applications through mapreduce and traffic analysis. In *Proceedings of the IEEE 18th International Conference on Parallel and Distributed Systems, ICPADS'2012*.

Wang, G., Butt, A., Pandey, P., and Gupta, K. (2009). A simulation approach to evaluating design decisions in mapreduce setups. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems. MASCOTS '09. IEEE International Symposium on*.

Yu, M., Greenberg, A., Maltz, D., Rexford, J., Yuan, L., Kandula, S., and Kim, C. (2011). Profiling network performance for multi-tier data center applications. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11.

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. (2010). Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10.