Escalonamento de processos sensível à localidade de dados em sistemas de arquivos distribuídos

Bruno Hott, Rodrigo Rocha, Dorgival Guedes

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG – Brazil

{brhott,rcor,dorgival}@dcc.ufmg.br

Resumo. *O crescimento acelerado dos dados disponíveis para pesquisas levou recentemente à popularidade de sistemas de processamento como Hadoop, Spark e outros, que se valem de sistemas de arquivos distribuídos para armazenar e acessar grandes volumes de dados. Para garantir um melhor desempenho, é importante que o processamento ocorra próximo aos dados, mas soluções atuais buscam localidade apenas no nível da rede local, não da mesma máquina. Este trabalho avalia o ganho de se colocar o processamento exatamente na máquina que contém os dados. Para isso, propomos um escalonador de tarefas sensível à localidade dos dados, integrado ao sistema de arquivos HDFS e ao escalonador YARN. Um protótipo foi implementado usando o ambiente Watershed e seu resultado comparado com outras políticas. Os resultados mostram que o escalonamento sensível ao posicionamento por máquina pode melhorar significativamente o desempenho de aplicações que utilizam o HDFS e mesmo aquelas que já se valem de uma solução que tenta garantir o acesso a dados já em memória, como o Tachyon.*

Abstract. *The accelerated growth of the volume of data available for researchers led recently to the increased popularity of distributed data processing systems like Hadoop, Spark and others. Those systems make use of distributed file systems to store, access and process big volumes of data. To guaratee better performance, it is important that the processing takes place close to the data. However, current solutions seek local network locality, not same machine locality. This work evaluates the gains of placing processing exactly in the same machine were data are. To achieve that we propose a data locality aware task placement scheduler integrated to the YARN scheduler and the HDFS file system. A prototype was implemented using the Watershed programming environment and its result is compared to other policies. Results show that scheduling based on machine-data placement can significantly improve performance of applications that use HDFS and even those that already use a memory-based data access solution, such as Tachyon.*

1. Introdução

O aumento da conectividade e da banda na Internet, combinados com a redução do custo de equipamentos eletrônicos em geral, têm causado uma explosão do volume de dados que trafegam pela rede. Ao mesmo tempo, recursos para armazenar esses dados vêm crescendo, o que levou ao surgimento de sistemas especialmente desenvolvidos para processá-los, tendo como um exemplo inicial o modelo MapReduce da Google [Dean and

Ghemawat 2008], que foi seguido por diversas implementações de código aberto, como Hadoop [White 2009], e novos modelos, como Spark [Zaharia et al. 2010b]. Além disso, tornou-se necessário uma solução para o armazenamento desse enorme conjunto de dados e sistemas de arquivos distribuídos, como HDFS [Shvachko et al. 2010] e Tachyon [Li et al. 2014], foram surgindo. Como os dados agora representam um volume muito grande e estão distribuídos por diversas máquinas em um cluster, surge o problema de levar as aplicações para perto das bases de dados de forma eficaz. Caso isso não seja feito, o preço de mover os dados pelo sistema pode ser muito alto e prejudicar o desempenho final da aplicação.

Apesar desse problema ser reconhecido, ainda existe pouco entendimento sobre a interferência da localidade dos dados no desempenho desses frameworks. Dependendo da localização, o acesso aos dados pela aplicação pode ser realizado diretamente no disco da máquina local, pela memória local, via caching, ou a partir da memória de outra máquina do cluster, via rede. Os diversos compromissos em termos de capacidade de armazenamento, tempo de acesso e custo computacional envolvidos tornam não trivial uma decisão de posicionamento. Por esses motivos, muitas vezes esses fatores podem ser melhor avaliados em tempo de execução da aplicação, quando se pode encontrar um melhor aproveitamento dos recursos do ambiente ou gastos com hardware.

Durante o trabalho de reengenharia do ambiente Watershed [Ramos et al. 2011, Rocha et al. 2016], realizamos a integração da nova implementação com o ecossistema Hadoop, utilizando o gerente de recursos YARN [Vavilapalli et al. 2013] e o sistema de arquivos HDFS. Em nossa experiência anterior com o Watershed e com seu antecessor, o Anthill [Ferreira et al. 2005], observamos sempre um impacto ao se dividir um fluxo de processamento de forma que o processamento inicial de dados lidos do disco fosse executado em um nó diferente daquele de onde os dados foram lidos. Com base nessa observação, decidimos implementar um escalonador sensível à localidade de dados, garantindo que a primeira tarefa de processamento definida para qualquer dado fosse sempre executada em uma máquina onde aqueles dados estivessem disponíveis localmente, seja em disco ou em memória.

Este trabalho avalia os aspectos de localidade em ambientes de processamento de dados massivos durante o desenvolvimento do escalonador. Com base na descrição da tarefa de processamento, que indica as etapas de processamento a serem executadas e o(s) arquivo(s) de entrada a ser(em) utilizado(s), das informações providas pelo HDFS (ou Tachyon) sobre a localização dos dados no sistema e dos recursos oferecidos pelo gerente de recursos YARN, nosso escalonador, sempre que possível, executa tarefas em máquinas onde aquele dado esteja acessível, seja em memória ou disco. Realizamos diversos experimentos com o intuito de comparar os resultados com outros escalonamentos possíveis. Os resultados obtidos comprovam as vantagens de se levar em conta o posicionamento dos dados no escalonamento de aplicações desse tipo.

Com isso em mente, o restante deste artigo está organizado da seguinte forma: a seção 2 fornece mais detalhes sobre os elementos do ecossistema Hadoop utilizados (HDFS, YARN e Tachyon), com base nos quais a seção 3 descreve a implementação do nosso escalonador, cujos resultados de avaliação são apresentados na seção 4. Finalmente, a seção 5 discute trabalhos relacionados e a seção 6 conclui com algumas observações finais e discussão de trabalhos futuros.

2. Elementos do ecossistema Hadoop

No desenvolvimento do nosso escalonador, três componentes do ecossistema Hadoop são de particular importância para o entendimento da solução adotada [Murthy et al. 2013]. Na seções a seguir, apresentamos os principais conceitos relacionados ao YARN, HDFS e Tachyon.

2.1. Serviço de gerenciamento de recursos e processos distribuídos

Idealmente, aplicações distribuídas decidem dinamicamente em qual nó de computação cada tarefa será executada, monitorando o estado de cada tarefa durante a execução. Monitorar o estado das tarefas possibilita, por exemplo, saber se as tarefas foram concluídas, bem como tomar medidas de tolerância às falhas sempre que necessário. Para ser capaz de realizar um escalonamento inteligente dessas tarefas distribuídas, considerando maior nível de detalhes do ambiente de execução, é necessário também gerenciar os recursos disponíveis oferecidos pelos nós de computação.

O YARN, escalonador de recursos do ecossistema Hadoop, oferece o serviço de uma plataforma básica para a gestão das aplicações distribuídas em um nível mais alto de abstração. Permite, assim, que diferentes aplicações distribuídas possam coexistir em um mesmo ambiente de execução em *cluster*. A unidade de alocação seria um *contêiner*, uma combinação de CPU, memória e disco, assinalada para executar o processamento de uma tarefa.

A principal ideia do YARN é particionar suas duas principais responsabilidades — gerenciamento de recursos e escalonamento de tarefas – em componentes separados: um gerenciador global de recursos (*ResourceManager* - RM) e uma tarefa mestre de gerenciamento por aplicação (*ApplicationMaster* - AM). O sistema YARN é composto pelo *ResourceManager* e uma tarefa escrava em cada nó do cluster, chamada de *NodeManager* (NM) [White 2009].

O *ApplicationMaster*, que é o módulo desenvolvido especificamente por cada aplicação, é o processo que coordenada a execução da própria aplicação no ambiente do *cluster*. O *NodeManager* é responsável por gerenciar a execução dos contêineres locais, monitorando suas utilizações de recursos — tais como CPU, memória, disco e rede — enviando para o RM informações sobre os contêineres. O *ResourceManager* é a autoridade máxima que arbitra a divisão dos recursos entre todos os aplicativos em execução no ambiente. O *NodeManager* tem um componente para o escalonamento de recursos, que é responsável pela alocação de recursos para os vários aplicativos em execução sujeito às restrições principais de capacidade, prioridade, e outros fatores. O escalonador não realiza nenhum monitoramento ou rastreamento para a aplicação, sem garantias de reiniciar tarefas que não são adequadamente finalizadas devido a qualquer falha da própria aplicação ou falhas de hardware. O escalonador executa seu escalonamento baseado em recursos requisitados pelo aplicativo usando a noção abstrata de um contêiner de recursos. A figura 1 ilustra o funcionamento do YARN no disparo de aplicações, cada uma com seu *Application Master*.

O escalonador discutido neste trabalho tira proveito do YARN para obter as informações sobre recursos disponíveis (nós de processamento), mas processa a informação de forma a garantir que os nós selecionados para processamento contenham

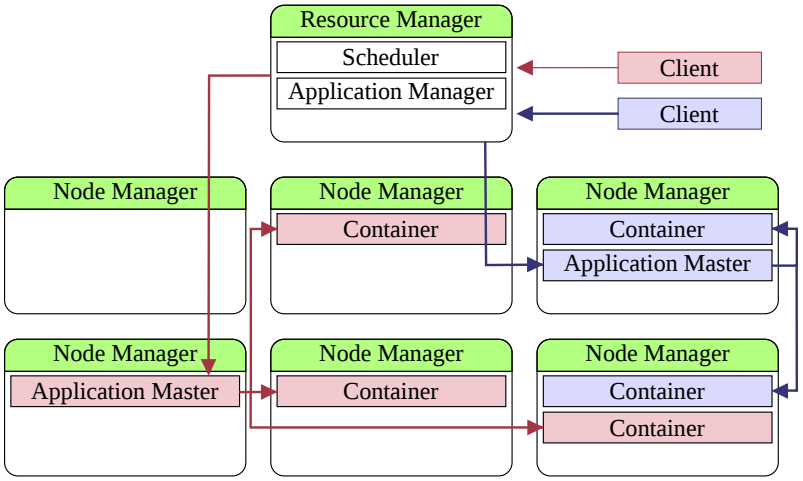


Figura 1. YARN gerenciando duas aplicações distintas em um ambiente de cluster.

os dados a serem processados. Isso é feito utilizando a informação fornecida pelos sistemas de arquivos HDFS/Tachyon (dependendo da configuração da aplicação). Ao disparar os processos da aplicação, cada processo é configurado com a informação necessária para que ele tenha acesso aos dados locais naquela máquina.

2.2. Sistema de arquivos distribuído do Hadoop

Sistemas para processamento de dados massivos lidam com coleções de dados que podem esgotar toda a capacidade de armazenamento de uma única máquina. Além disso, processar tais coleções de dados demanda grande poder computacional. Por esse motivo, grandes coleções de dados são armazenadas de maneira distribuída em diversos nós computacionais, permitindo que grandes volumes de dados sejam armazenados, além da capacidade individual de cada máquina, bem como o processamento e acesso distribuído de tais coleções de dados.

O *Hadoop Distributed File System* (HDFS) [Shvachko et al. 2010] foi baseado principalmente no *Google File System* (GFS) [Ghemawat et al. 2003], que é um sistema distribuído de arquivos que proporciona tolerância às falhas mesmo enquanto executando sobre hardware considerados como produtos de conveniência. Nesse sistema, arquivos são escritos sequencialmente quando criados, ou dados são acrescentados sequencialmente ao final de um arquivo já existente, pela operação de *append*[1].

O HDFS é responsável por armazenar metadados sobre o sistema de arquivo bem como arquivos de usuários. Arquivos de metadados são armazenados em um servidor dedicado, chamado de *NameNode*, enquanto os arquivos de usuários são armazenados em outros servidores chamados de *DataNodes*. Todos os arquivos têm seus dados divididos em blocos grandes, que são distribuídos pelos discos dos *DataNodes*. Cada bloco é também replicado em outros *DataNodes* por motivos de confiabilidade na recuperação da informação. Além de garantir durabilidade dos dados, esta estratégia tem a vantagem adicional de que a largura de banda para transferência de dados é ampliada, e há mais

[1]As primeiras versões do HDFS não permitiam a operação de *append*, porém essa funcionalidade foi adicionada em versões posteriores (versão `0.20-append` ou `0.20.2`)

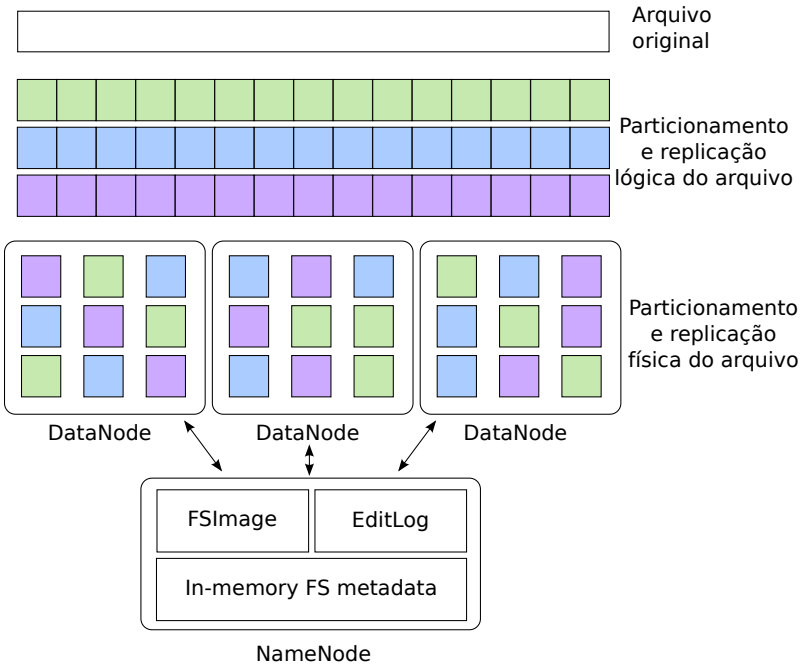


Figura 2. Visão geral da organização do HDFS, ilustrando o particionamento e replicação distribuída de um arquivo.

oportunidades para escalonar a computação para perto dos dados necessários. A figura 2 ilustra como um arquivo é particionado em blocos de tamanhos iguais (geralmente 64 ou 128 MB), de maneira que cada bloco é replicado nas diferentes máquinas que compõem o cluster.

Todos os servidores são totalmente conectados e se comunicam uns com os outros por meio de protocolos baseados em TCP. Para realizar operações de leitura ou escrita de dados, um cliente requisita ao *NameNode* quais *DataNodes* devem ser acessados e, em seguida, interage diretamente com cada *DataNode*, nunca realizando tais operações por intermédio do *NameNode*.

Quando um arquivo mantido no HDFS é identificado como parte de uma aplicação, o escalonador consulta o HDFS para obter a lista de blocos daquele arquivo. Essa lista indica, para cada bloco, quais nós possuem réplicas do mesmo. De posse dessa informação, o escalonador faz a associação de nós de processamento com os blocos do HDFS que eles devem processar.

2.3. Tachyon

Os autores do Tachyon [Li et al. 2014] argumentam que os conjuntos de dados de interesse a qualquer instante (*working set*) na maioria dos datacenters são relativamente pequenos se comparados ao conjunto total de dados neles existentes.

Frameworks existentes armazenam dados intermediários em memória com a tarefa (job) e armazenam os dados de entrada como cache em memória. Porém uma operação permanece sem executar em memória, que é o compartilhamento de dados entre as tarefas. Em particular, armazenar a saída de uma tarefa com segurança com a intenção de compartilhá-la com outras tarefas é um processo lento, já que o dado é replicado através

da rede e discos para tolerância às falhas. Isto faz com que os sistemas de armazenamento em cluster atuais sejam ordens de magnitude mais lentos do que escrever em memória. Esta é a principal motivação dos criadores do Tachyon. Nesse sentido, aquele sistema foi desenvolvido para oferecer um acesso eficiente aos dados de interesse em um certo momento, mantendo-os em memória, mas com uma forma eficiente de acessar o disco para leitura e escrita, quando necessário.

Do ponto de vista estrutural, Tachyon utiliza uma arquitetura padrão de mestre-escravo similar ao HDFS, onde cada *worker* gerencia blocos locais e os compartilha com as aplicações através da memória local.

Cada *worker* executa um daemon que gerencia os recursos locais e reporta periodicamente seu status para o master. Além disso, cada *worker* usa a RAM para armazenar arquivos mapeados em memória. Dessa forma uma aplicação com localidade de dados pode interagir à velocidade da memória.

Do ponto de vista do nosso trabalho, Tachyon pode ser visto como uma cache com características semânticas especializadas para lidar com sistemas de arquivos distribuídos. O nosso escalonador foi desenvolvido para também interfacear com esse sistema, a fim de poder explorar não só a localidade em disco, mas também aquela em memória, potencialmente agregando o benefício da localidade ao acesso em memória local, ao invés de exigir uma transferência pela rede.

3. Implementação

A execução de uma aplicação dentro do ecossistema Hadoop, seja ela baseada no Hadoop MapReduce, Spark ou outros ambientes de programação, implica na definição de pelo menos duas informações principais: a identificação dos dados de entrada a serem processados (p.ex., o arquivo de entrada) e das tarefas que devem ser executadas (inclusive quantas instâncias de cada tarefa devem existir).

De posse dessas informações, o escalonador proposto no presente trabalho realiza uma consulta ao sistema de arquivos distribuído (HDFS/Tachyon) para obter a localização dos blocos (incluindo todas as réplicas) que armazenam porções dos dados de entrada. Os nós identificados são posteriormente ordenados pela quantidade de blocos existentes em cada um, de forma a priorizar os nós do cluster com a maior quantidade de blocos.

O processo de escalonamento dos blocos é dividido em duas fases, sendo a primeira responsável pela escolha das porções do arquivo que serão lidos localmente por cada instância. Para isso, são selecionados até b/i blocos locais às máquinas onde as instâncias serão executadas, onde b é o número total de blocos a serem lidos e i é o número de instâncias, considerando inclusive que podem haver mais instâncias em uma mesma máquina. Esses conjuntos são então distribuídos entre as instâncias da aplicação.

Na segunda fase, realizamos o escalonamento dos blocos que serão lidos remotamente por cada instância. Como o HDFS não garante a distribuição uniforme dos arquivos armazenados, geralmente não é possivel que toda leitura seja realizada localmente sem que o balanceamento da aplicação seja prejudicado. Por esse motivo, o escalonador verifica se faltam blocos para serem escalonados nos nós e, caso houverem, seleciona aqueles que não foram escolhidos na primeira fase e que serão, então, lidos remotamente. Esse processo é repetido até que todas as instâncias tenham recebido todos os blocos

devidamente.

O escalonador então utiliza um mecanismo de requisição sem relaxamento na localidade dos contêineres YARN, forçando que os mesmos sejam alocados considerando exatamente as localizações especificadas, derivadas das informações obtidas das fontes de dados (HDFS/Tachyon).

O Tachyon possui um funcionamento conceitualmente equivalente ao HDFS, porém os blocos de um dado arquivo podem em qualquer momento ser encontrados em blocos no disco ou já carregados na memória do sistema. Dessa forma, o escalonador prioriza nós que contenham blocos do arquivo em memória principal. Caso esses blocos em memória não existam, então a localização dos dados em disco é considerada.

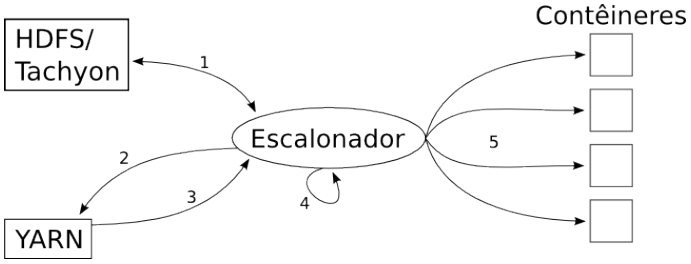


Figura 3. Passos de operação do escalonador: (1) busca do arquivo no HDFS/-Tachyon com lista de blocos, (2) pedido dos contêineres, (3) recebimento dos contêineres fora de ordem, (4) organização dos contêineres em função dos blocos e (5) execução dos processos da aplicação.

A figura 3 ilustra os passos de operação do escalonador desenvolvido. Primeiramente, o escalonador acessa o sistema de arquivo e obtém as informações sobre os arquivos envolvidos, contendo a lista de blocos e a identificação da localização de cada réplica dos mesmos. De posse dessas informações, é possível solicitar ao YARN os contêineres nos locais exatos desejados. Pela forma de operação do YARN, os contêineres para as solicitações enviadas podem ser retornados fora de ordem, então o escalonador deve processar cada resposta para identificar o nó obtido e associá-lo aos blocos presentes naquele nó. Finalmente, os processos podem ser disparados nos diversos nós com a informação sobre quais blocos eles devem acessar (que serão sempre blocos locais).

Utilizamos essa heurística de escalonamento para mantermos maior controle sobre a alocação das instâncias em relação aos seus respectivos dados de entrada e assim melhor avaliarmos o impacto da localidade de dados. Como trabalhos futuros pretendemos utilizar novas heurísticas que também levem em conta outros aspectos como a carga de trabalho atual de cada máquina.

4. Avaliação experimental

Para avaliar o desempenho do escalonador, executamos uma aplicação que lia os dados de um arquivo de entrada e computava estatísticas simples sobre o mesmo. Além do escalonador proposto, a aplicação foi executada ainda com 3 outras políticas de escalonamento: sem nenhum escalonador além do que já é provido pelo YARN, com um escalonamento aleatório e com um escalonamento que buscava representar o pior caso, onde cada processo sempre era executado em uma máquina diferente daquela que continha os dados a

serem processados. Nas figuras a seguir, os resultados do nosso escalonador são identificados pela sigla "esc", enquanto as demais situações são identificadas pelas siglas "sec", "ale" e "pes", respectivamente.

Os testes foram executados em um cluster com 10 máquinas virtuais (VMs), onde cada uma delas possuía 2 VCPUs de 2,5 GHz e 4 GB de memória RAM. Cada máquina virtual foi associada a um disco na própria máquina física em que a VM foi criada, para garantir o controle preciso de localidade física dos blocos. Todos os testes foram executados cinco vezes. Os tempos de execução apresentados se referem à média das execuções; o desvio padrão de cada conjunto de execuções é sempre apresentado nas figuras.

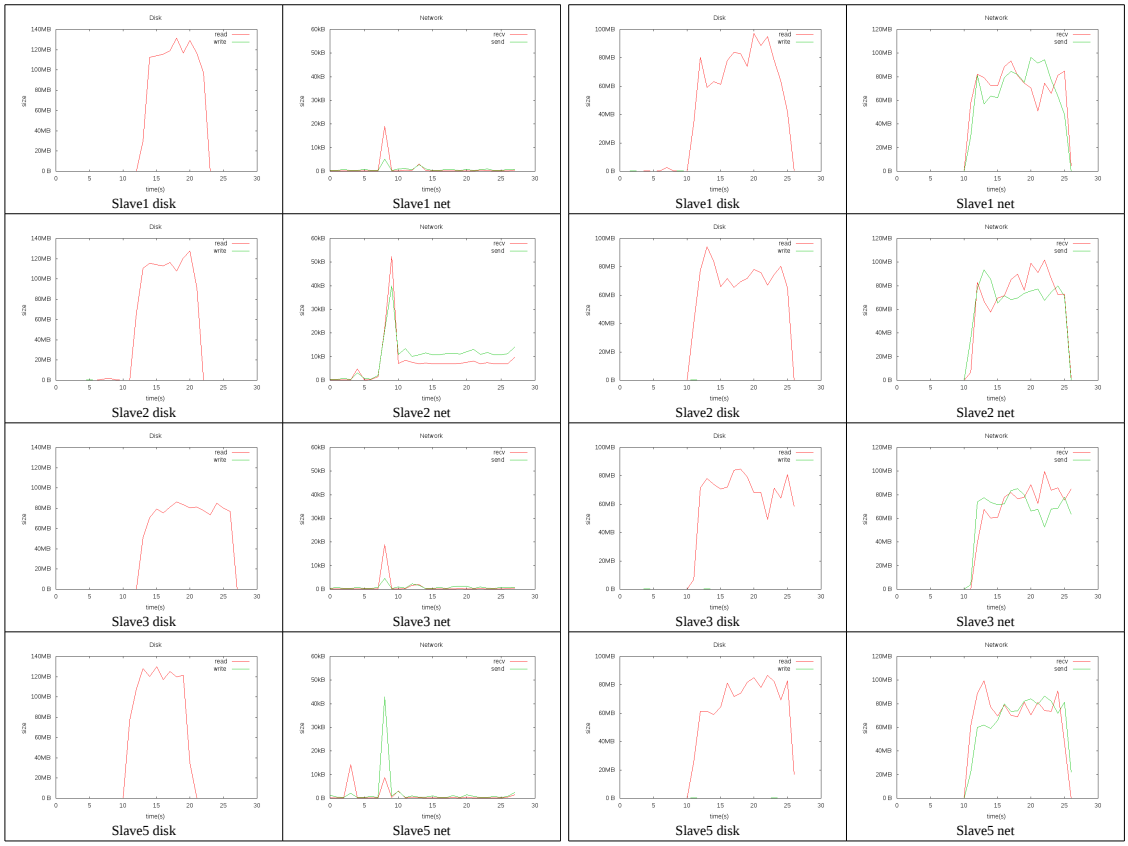
Para confirmar se o escalonador estava operando corretamente, utilizamos o comando *dstat* para coletar dados sobre o tráfego de disco e de rede em cada máquina durante os testes. Os dados coletados para essas duas grandezas comprovam que, para o escalonador proposto, os nós apresentavam um acesso a disco condizente com os dados locais a serem lidos e não havia tráfego de rede significativo, além de operações de sincronização no ambiente de processamento. Já no caso da política que sempre assinalava um processo para executar em um nó onde os dados não estavam, os dados mostravam que o padrão de acesso a discos era o mesmo do tráfego de rede enviado pelo nó — isto é, todas as leituras de disco eram destinadas a atender um processo em outro nó. Ao mesmo tempo, o padrão de dados lidos pela rede correspondia ao padrão de leitura de disco na máquina que continha os dados exigidos pelo processo da aplicação executando naquele nó. A figura 4 ilustra uma execução com quatro nós para os dois casos.

Para avaliar o comportamento do escalonamento sensível à localidade dos dados, executamos o programa de leitura de arquivos em diversas situações e medimos o tempo de execução total. Dois elementos importantes na análise foram o tamanho do cluster considerado e o tamanho do bloco usado no sistema de arquivos distribuído. O tamanho do arquivo usado era determinado em termos de número de blocos, então experimentos com tamanho de blocos maiores liam arquivos proporcionalmente maiores. Todos os arquivos foram gerados com fator de replicação 2.

Nos testes utilizando o HDFS, para que a cache do sistema de arquivos do sistema operacional não interferisse nos experimentos, efetuamos a limpeza de cache antes de cada execução. Nos testes usando o Tachyon, como seu princípio de operação é exatamente o de tentar se aproveitar ao máximo de todas as possibilidades de cacheamento dos dados, a cache era limpa apenas antes do início de cada experimento, a aplicação era executada uma primeira vez para carregar a cache do Tachyon e depois era medido o conjunto de cinco execuções.

A figura 5 mostra o efeito da variação do tamanho do bloco para dois tamanhos de cluster diferentes (4 e 8 máquinas). Tempos de execução para o cluster maior são sempre superiores, pois há mais overhead na inicialização e na sincronização entre os processos na aplicação distribuída. Claramente, o escalonador proposto apresenta bom desempenho em todos os casos.

Para o cluster de 4 máquinas, praticamente não há diferença entre os tempos do escalonador usual do YARN e do escalonador baseado em localidade, exceto para blocos de 1GB. Observando os logs da execução, um dos motivos é que, com 4 máquinas e fator de replicação 2, o próprio escalonador do YARN já balanceava os recursos de forma



(a) Execução com o escalonador proposto (b) Execução com o escalonador sem localidade

Figura 4. Padrões de acesso a disco e de dados enviados e recebidos pela rede para uma execução de uma aplicação com quatro nós de processamento, com o escalonador sensível à localidade dos dados (a) e com uma política de escalonamento que sempre posiciona o processamento em um nó diferente daquele que contém os dados (b). Pode-se ver que no primeiro caso não há tráfego de rede significativo, enquanto no segundo há sempre um volume de tráfego enviado semelhante ao padrão de leitura local.

satisfatória e tinha pelo menos 50% de chance de alocar um processador junto ao dado que ele iria processar. Já o escalonador aleatório, teve desempenho mais próximo ao pior caso por não levar em conta as informações sobre a carga dos nós que era considerada pelo YARN.

Já para o cluster de 8 máquinas, a chance de um processo ser escalonado aleatoriamente em um nó com o dado associado a ele cai para 25% e o escalonador sensível à localidade dos dados passa a ter um desempenho sensivelmente melhor, o que comprova que executar o processamento inicial dos dados na mesma máquina que os contém ainda é vantajoso em relação ao custo de acesso pela rede. Além disso, mesmo para o cluster menor, no caso de blocos de leitura/arquivos maiores, o ganho de processar os dados localmente também é visível. Na verdade, o maior ganho do escalonador foi exatamente para o cluster de 4 máquinas e blocos de 1 GB: aproximadamente 20% mais rápido que o escalonador default do YARN e duas vezes mais rápido que o pior caso. Já no cluster de 8 máquinas, o ganho de aproximadamente 20% em relação ao YARN permanece, mas os custos de sincronização com mais máquinas reduzem o ganho em relação ao pior caso.

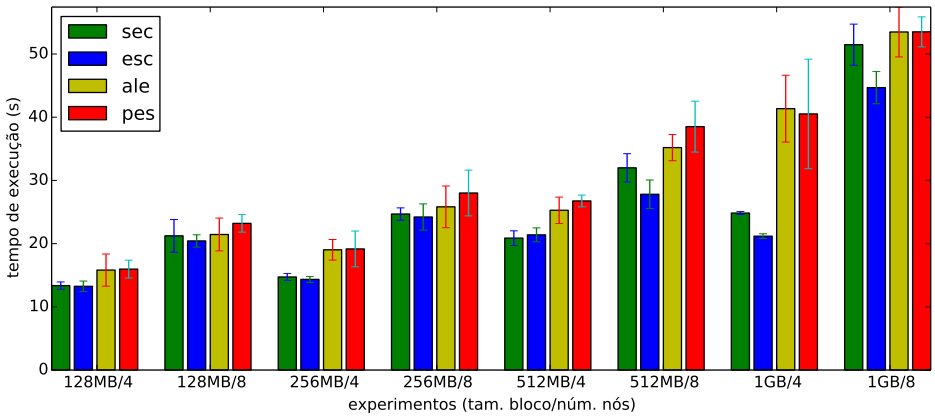


Figura 5. Resultados experimentais utilizando o escalonar integrado ao HDFS para diferentes tamanhos de blocos, diferentes números de máquinas.

Recentemente, diversos trabalhos têm sugerido a adoção de mecanismos de armazenamento em memória, como o projeto RamCloud [Ousterhout et al. 2010]. Nesse caso, o tempo de acesso e as taxas de transmissão de dados seriam aqueles da memória e não do disco, o que pode alterar significativamente o desempenho do sistema. Considerando esse cenário, o escalonador também foi configurado para trabalhar com o Tachyon, já que esse oferece o recurso de manter em memória arquivos já acessados. Como mencionado anteriormente, para os experimentos executávamos a aplicação uma primeira vez, causando a carga do arquivo para a memória do Tachyon e medíamos os tempos de cinco execuções depois que os dados já se encontravam na memória de algum nó. Devido ao comportamento do Tachyon e da aplicação, não havia replicação de blocos em memória: cada bloco era lido por apenas um processo parte da aplicação, então o dado era carregado na memória de apenas uma máquina de cada vez.

A figura 6 mostra os resultados nesse caso (não foi utilizado o tamanho de bloco de 1GB devido a limitações do Tachyon na configuração usada nos clusters do experimento). Claramente, nesse caso os ganhos do escalonador baseado em localidade são ainda mais significativos. Já que não há replicação de blocos nos dados em memória, o escalonador do YARN já não é capaz de obter o mesmo desempenho na maior parte dos casos. Além disso, o fato dos dados já estarem na memória do nó local tornam o tempo de execução bem mais regular (e menor) para o escalonador proposto. Apesar do aumento do número de máquinas do cluster ainda ter um impacto, os tempos de execução ainda caem significativamente devido às taxas de transferências mais altas.

Com o Tachyon, o escalonamento proposto é pelo menos 25% melhor que o escalonador original do YARN. Por outro lado, para o maior cluster e maior tamanho de bloco considerados, o escalonamento baseado em localidade chega a uma redução de mais de 50% em relação ao YARN isoladamente e mais de 65% em relação ao pior caso. Esses resultados indicam que, em um cenário baseado em memória, a noção de processamento dos dados na origem se torna ainda mais importante, dadas as tecnologias de rede atual.

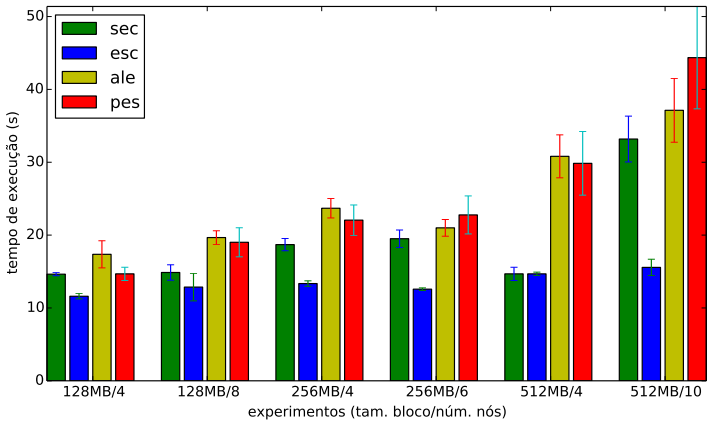


Figura 6. Resultados experimentais utilizando o escalonar integrado ao Tachyon para diferentes tamanhos de blocos, diferentes números de máquinas.

5. Trabalhos relacionados

Os artigos que descrevem Hadoop, Spark, HDFS (e GFS) e Tachyon, já mencionados, discutem alguns dos aspectos de decisão envolvidos na escolha de nós de processamento ou armazenamento em cada caso. Em geral, entretanto, o princípio de escalonamento de tarefas é o padrão do YARN, que se baseia em uma descrição simples do ambiente em termos de máquinas e racks [White 2009]. Nesse caso, processos são executados preferencialmente em uma máquina no mesmo rack onde os dados estão, mas não há um esforço maior em localizar processamento no mesmo nó que contém os dados, como neste trabalho.

Um dos primeiros trabalhos a discutir o impacto da localidade no processamento de grandes volumes de dados no contexto de datacenters modernos é devido a Barroso e Hölzle [Barroso and Hölzle 2009]. Nele os autores chamam a atenção para as diferenças em termos de acesso entre discos e memória RAM em uma mesma máquina, em um mesmo rack ou em racks diferentes. Essas diferenças, que se tornam mais significativas entre máquinas em racks diferentes, foram a base para decisões de projeto com as do Hadoop. Entretanto, os autores mostram que ainda há um degrau de desempenho significativo ao se optar por máquinas diferentes, mesmo que no mesmo rack (ligadas por um único switch de rede).

Na literatura, temos alguns escalonadores que tem como objetivo utilizar eficientemente os recursos de ambientes multiusuário. O HaSTE [Yao et al. 2014] é um escalonador integrado ao YARN, desenvolvido especificamente para aplicações MapReduce, cujo foco principal é reduzir o tempo total de conclusão de múltiplas aplicações em execução concorrente, além de também reduzir a utilização de recursos do cluster. Esse objetivo é alcançado considerando os recursos requisitados, a capacidade geral do cluster e as dependências entre as tarefas de cada aplicação. Ainda para ambientes multiusuário, Zaharia et al. [Zaharia et al. 2010a] apresentam uma técnica de escalonamento centrado em localidade de dados, chamada de *delay scheduling*, também desenvolvendo um escalonador para tarefas MapReduce no ambiente Hadoop. Os autores discutem uma heurística onde as tarefas são adicionadas à uma fila enquanto as máquinas que armazenam os da-

dos necessários estiverem ocupadas. Essa abordagem contribui para o uso equilibrado dos recursos entre os usuários ao mesmo tempo que visa localidade dos dados.

Isard et al. [Isard et al. 2009] apresentam um escalonador chamado Quincy para a plataforma distribuída Dryad [Isard et al. 2007]. O Quincy mapeia o problema de escalonamento como um grafo de fluxos, onde as arestas representam os diversos custos envolvidos no escalonamento de cada tarefa entre os diversos nós disponíveis no cluster, considerando aspectos como localidade de dados e balanceamento de carga de trabalho entre os nós.

Ousterhout et al. [Ousterhout et al. 2010] advogam o uso de sistemas de arquivos em RAM (Ramclouds) para evitar o custo do acesso a disco. Nesse sentido, aquele trabalho é ortogonal ao apresentado aqui, já que nosso objetivo é levar o processamento para a mesma máquina onde o dado se encontra, seja em memória ou disco. Os experimentos como Tachyon mostram que essa exploração do conceito de localidade pode ser até mais benéfico se o dado já se encontra em memória.

Em uma primeira análise, este trabalho se contrapõe ao trabalho de Ananthanarayanan et al. [Ananthanarayanan et al. 2011], que vai contra a preocupação com localidade de discos. Naquele artigo, os autores argumentam que diversos avanços em curso poderão vir a tornar a localidade de disco irrelevante. Nosso argumento neste artigo não tem por objetivo contrariar aquele nessa previsão. Apenas mostramos que, em condições atuais, a localidade ainda é significativa. Além disso, mostramos também que em um contexto com dados armazenados em memória, o escalonamento baseado na localidade dos dados em memória também pode resultar em ganhos de desempenho.

6. Conclusão

A popularidade do ecossistema Hadoop para o processamento de dados massivos (*big-data*) levou ao desenvolvimento de diversos sistemas de processamento baseados no sistema de arquivos HDFS e no escalonador YARN. Esses ambientes utilizam diretamente as políticas de gerência de espaço de armazenamento e escalonamento propostas pelo Hadoop, que são baseadas na noção de localidade de rack (máquinas interligadas diretamente por apenas um switch de rede).

Neste trabalho avaliamos o impacto dessa escolha comparada a uma política de alocação de recursos sensível à localidade dos dados em termos de máquinas. O argumento nesse caso é que, pelo menos para a tecnologia de Ethernet Gigabit atualmente comum na maioria dos datacenters e clusters, processar dados em uma máquina que não aquela que já os mantém gera um overhead de comunicação pela rede, mesmo que não haja gargalos entre fonte e destino. Para avaliar esse argumento, desenvolvemos um escalonador sensível à localidade dos dados integrado ao HDFS e ao YARN durante a reengenharia do ambiente de processamento Watershed.

Os resultados comprovam que o escalonamento sensível à localidade não prejudica o desempenho do sistema em clusters pequenos e com blocos menores e beneficia significativamente a execução em clusters maiores e quando os blocos de dados são maiores. Além disso, os ganhos se tornam ainda mais significativos quando os dados já estão em memória, como no caso do Tachyon. Isso pode ser um resultado ainda mais importante considerando-se propostas recentes de sistemas de armazenamento baseados em memória.

O escalonador proposto é facilmente extensível e pode ser integrado a outros ambientes de processamento que usam o ecossistema Hadoop. Como trabalhos futuros, pretendemos realizar a inclusão do mesmo no Hadoop MapReduce e no ambiente Spark, além de estender a política de escalonamento para incluir heurísticas para casos de alocação em cenários multiusuários com maior contenção de recursos.

Agradecimentos

Este trabalho foi parcialmente financiado por Fapemig, CAPES, CNPq, e pelos projetos MCT/CNPq-InWeb (573871/2008-6), FAPEMIG-PRONEX-MASWeb (APQ-01400-14), e H2020-EUB-2015 EUBra-BIGSEA (EU GA 690116, MCT/RNP/CETIC/Brazil 0650/04).

Referências

Ananthanarayanan, G., Ghodsi, A., Shenker, S., and Stoica, I. (2011). Disk-locality in datacenter computing considered irrelevant. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, pages 1–5, Berkeley, CA, USA. USENIX Association.

Barroso, L. A. and Hölzle, U. (2009). The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108.

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

Ferreira, R. A., Meira, W., Guedes, D., Drummond, L. M. d. A., Coutinho, B., Teodoro, G., Tavares, T., Araujo, R., and Ferreira, G. T. (2005). Anthill: A scalable run-time environment for data mining applications. In *Computer Architecture and High Performance Computing, 2005. SBAC-PAD 2005. 17th International Symposium on*, pages 159–166. IEEE.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 29–43, New York, NY, USA. ACM.

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2007 Eurosys Conference*, Lisbon, Portugal. Association for Computing Machinery, Inc.

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*, pages 261–276. ACM.

Li, H., Ghodsi, A., Zaharia, M., Shenker, S., and Stoica, I. (2014). Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 6:1–6:15, New York, NY, USA. ACM.

Murthy, A., Vavilapalli, V. K., Eadline, D., Markham, J., and Niemiec, J. (2013). *Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2*. Pearson Education.

Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., Rumble, S. M., Stratmann, E., and Stutsman, R. (2010). The case for ramclouds: Scalable high-performance storage entirely in dram. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105.

Ramos, T., Silva, R., Carvalho, A. P., Ferreira, R. A. C., and Meira, W. (2011). Watershed: A high performance distributed stream processing system. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2011 23rd International Symposium on*, pages 191–198. IEEE.

Rocha, R., Hott, B., Dias, V., Ferreira, R., Meira, W., and Guedes, D. (2016). Watershedng: an extensible distributed stream processing framework. *Concurrency and Computation: Practice and Experience*. (accepted for publication). doi:10.1002/cpe.3779.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA. IEEE Computer Society.

Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., et al. (2013). Apache hadoop YARN: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM.

White, T. (2009). *Hadoop: the definitive guide: the definitive guide*. O'Reilly Media, Inc.

Yao, Y., Wang, J., Sheng, B., Lin, J., and Mi, N. (2014). Haste: Hadoop yarn scheduling based on task-dependency and resource-demand. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 184–191. IEEE.

Zaharia, M., Borthakur, D., Sen Sarma, J., Elmeleegy, K., Shenker, S., and Stoica, I. (2010a). 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, pages 265–278, New York, NY, USA. ACM.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010b). Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10.