Análise distribuída em dados meteorológicos utilizando o modelo de atores

Jimmy K. M. Valverde-Sánchez, Otávio Carvalho, Eduardo Roloff, Nicolas Maillard, Philippe O. A. Navaux

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{jkmvsanchez,omcarvalho,eroloff,nicolas,navaux}@inf.ufrgs.br

Abstract. *The GRIB scientific data format is widely used in the meteorological community to store weather and weather forecast data. These datasets are shared among the meteorological centers and used as an input to the weather models, both for weather forecasting and historical analysis. However, the current methods for processing files in this format no perform the computation in a distributed environment. This situation limits the analytical capabilities of scientists who need to perform analysis on large data sets in order to obtain information in the shortest time possible using of all available resources. In this sense, this work presents an alternative to data processing in this format, using the Actor model implemented in Akka toolkit, evaluate our proposal in the Cloud and compare it with other techniques.*

Resumo. *No âmbito da meteorologia o formato de dados científicos GRIB é amplamente utilizado para armazenar histórico de dados e previsões meteorológicas. Esses conjuntos de dados são compartilhados entre os centros meteorológicos e usados como uma entrada para os modelos de clima, tanto para a previsão do tempo e análise histórica. No entanto, as ferramentas atuais disponíveis e métodos para processar arquivos neste formato não realizam o processamento em um ambiente distribuído. Essa situação limita as capacidades de análise dos cientistas que precisam realizar uma análise sobre grandes conjuntos de dados com o objetivo de obter informação no menor tempo possível fazendo uso de todos os recursos disponíveis. Neste contexto, este trabalho apresenta uma alternativa ao processamento de dados nesse formato fazendo uso do modelo de atores implementado no Akka toolkit, avaliamos a nossa proposta na nuvem e a comparamos com outras técnicas.*

1. Introdução

Estamos vivendo na era de Big Data [Dobre and Xhafa 2014], que é resultado do massivo aumento nas quantidades de dados geradas, em intervalos de tempo cada vez menores, através de diversas fontes, tais como *smartphones*, sensores, simulações e *logs* de aplicações. O processamento dessas quantidades massivas de dados é um desafio não somente para as corporações, mas também para a comunidade científica, que gera quantidades massivas de dados em seus experimentos. Para que possamos realizar o processamento eficiente dessa nova gama de perfis de dados, necessitamos contar não somente com maiores quantidades de recursos computacionais, mas também utilizarmos melhores práticas de processamento.

Portanto, é cada vez mais necessário desenvolvermos aplicações capazes de extrair resultados válidos da informação contida nesses grandes conjuntos de dados. Na comunidade científica, podemos utilizar o conhecimento extraído desses dados com o objetivo de compreender fenômenos em áreas chaves do conhecimento, como meteorologia, sismologia e saúde.

O modelo de programação sequencial já não é suficiente para lidar com os requisitos destes grandes conjuntos de dados. Para que possamos extrair conhecimento desses perfis de dados, modelos de programação paralela e distribuída tornam-se valiosos para extrairmos informações em tempos de processamento razoáveis [Kambatla et al. 2014].

Os esforços para a análise e processamento de grandes conjuntos de dados, em ambientes paralelos e distribuídos tem sido focados principalmente em formatos de dados científicos específicos, tais como o NetCDF [Li et al. 2003], [Zhao et al. 2010], [Buck et al. 2011], [Wang et al. 2012] e o HDF5 [Wang et al. 2012], [Blanas et al. 2014]. Por outro lado, existem formatos de dados, amplamente utilizados pela comunidade científica, para os quais ainda não existem soluções explorando um ambiente paralelo e distribuído. Um exemplo disso é o formato GRIB[1], que é amplamente utilizado no âmbito da comunidade meteorológica, para armazenar e compartilhar o histórico de dados e previsões meteorológicas. Esse formato de dados, ao contrário de formatos com o NetCDF e HDF5, não suporta acesso aleatório aos dados [Fortner 1995]. Dessa forma, o processamento paralelo de larga escala, para esse tipo de arquivos, ainda é um desafio de pesquisa pouco explorado, carecendo de metodologias e ferramental específico.

O Centro de Previsão de Tempo e Estudos Climáticos (CPTEC[2]) é uma situação real onde o processamento de arquivos meteorológicos em formato GRIB é fundamental para a comunidade científica. Neste centro de pesquisa são executados periodicamente diversos modelos de previsões meteorológicas, abrangendo toda a América do Sul e os oceanos adjacentes. Os resultados dessas previsões e a condição inicial do modelo são fornecidas duas vezes por dia, utilizando o formato GRIB. Na fase de pré-processamento da execução do BRAMS[3], modelo de previsão numérica do tempo usado para simular condições atmosféricas, os arquivos GRIB devem ser convertidos em um arquivo intermediário que contém os dados de entrada para o modelo. O processamento é realizado usando um modelo sequencial, o qual leva um tempo considerável do processo global.

Neste trabalho, propomos uma maneira alternativa de processar e realizar a análise de grandes quantidades de dados no formato GRIB, realizando processamento distribuído através do modelo de atores, implementado no projeto Akka[4]. O Akka vem sido amplamente utilizada pela comunidade científica, principalmente através de projetos como Apache Spark e Flink, que usam o Akka internamente para sua comunicação distribuída. No entanto, existem limitações para conjuntos de dados que excedem largamente as capacidades de memória. Quando isso acontece no Spark, por exemplo, é necessário armazenar as abstrações de dados distribuídos (RDDs[5]) em disco, o que causa perda de desempenho ou pode acarretar até mesmo no lançamento de exceções *OutOfMemory* [Gu and Li 2013].

[1]http://www.nco.ncep.noaa.gov/pmb/docs/on388/
[2]http://www.cptec.inpe.br/
[3]http://brams.cptec.inpe.br
[4]http://akka.io/
[5]http://spark.apache.org/docs/latest/programming-guide.html

2. Motivação

Os atuais métodos para processamento de arquivos no formato GRIB (GRIB API[6], wgrib2[7]) apresentam uma carência muito relevante: A possibilidade de realizar o processamento utilizando mais de um servidor, de maneira distribuída. Dessa forma, a escalabilidade do processamento é limitada ao poder de processamento de apenas um único nó computacional.

A motivação desse trabalho é propor o uso de ferramentas que possibilitem explorar o processamento de maneira distribuída. Ao realizarmos o processamento de maneira distribuída, dois benefícios são alcançados. Primeiro, desenvolve-se a possibilidade de processamento de uma maior quantidade de arquivos de uma única vez. Segundo, através do processamento distribuído, existe a possibilidade de redução do tempo total de processamento através da divisão de tarefas entre os nós de processamento.

3. Conceitos Básicos

Nesta seção são apresentados conceitos úteis para obtermos um melhor entendimento das seções posteriores do artigo.

3.1. GRIB

O GRIB(GRIdded Binary) é um formato binário de dados científicos, criado pela Organização Meteorológica Mundial (WMO)[8], principalmente com o objetivo de transmitir e armazenar dados meteorológicos [WMO 2003]. Este formato é amplamente utilizado na comunidade meteorológica para armazenar dados históricos e de previsões meteorológicas [Markiewicz et al. 2013]. Contudo, este formato não é facilmente acessível [Candanedo et al. 2013], ao contrário de formatos similares como o NetCDF [Rew and Davis 1990] e o HDF5 [The HFD Group 2016], por não permitir acesso aleatório aos dados.

Ao longo do tempo, a versão 0 do formato tornou-se obsoleta, tendo sido substituída pela versão 1. Após certo tempo, foi proposta a versão 2 do formato, afim de permitir a representação de parâmetros não cobertos na versão 1 que eram necessários para os centros meteorológicos. A definição desses parâmetros e algumas informações dentro de uma mensagem GRIB são parte do padrão, mas não da mensagem, a qual armazena uma referência às informações definidas em uma tabela externa.

Cada arquivo pode conter uma ou várias mensagens e cada mensagem GRIB contém divisões lógicas chamadas seções, concretamente na versão 2 existem 9 seções. A primeira seção, *Indicator*, marca o início de cada mensagem e contém a Disciplina (a disciplina 0, por exemplo, corresponde a produtos meteorológicos) dos dados na mensagem, a versão e o tamanho total da mensagem em bytes. As demais seções, exceto a última, contém o tamanho da mensagem e o número da seção. A seção *Identification* define as características que se aplicam ao conteúdo da mensagem. A seção *Local use* é opcional e define itens adicionais que são próprios do centro meteorológico de origem. A seção *Grid definition* define a superfície da grade e a geometria dos valores na superfície (por

[6]https://software.ecmwf.int/wiki/display/GRIB/Home
[7]http://www.cpc.ncep.noaa.gov/products/wesley/wgrib2/index.html
[8]https://www.wmo.int/

exemplo, Latitude-longitude, Mercator). A seção *Product definition* contém a natureza dos dados. A seção *Data representation* define o método de empacotamento dos dados usado e a informação necessária para a decodificação dos mesmos (por exemplo, *Complex Packing and Spatial Differencing*). A seção *Bit-map* indica a presença ou a ausência de dados em cada ponto da grade. A seção *Data* contém os valores para cada ponto da grade e, finalmente a seção *End* indica o fim da mensagem com o código 7777.

3.2. Akka toolkit

O Akka visa facilitar o desenvolvimento de aplicações concorrentes e distribuídas, tendo sido baseado no Modelo de Atores proposto em [Hewitt et al. 1973]. Ele é baseado em entidades autônomas, denominadas Atores, que se comunicam com outros atores através de mensagens assíncronas. Cada ator possui uma caixa de correio, onde as mensagens recebidas são armazenadas até serem processadas. Os atores também possuem um comportamento que define como as mensagens recebidas devem ser processadas, os atores não compartilham estados, comunicando-se somente através de mensagens. Através dessas características, o Akka toolkit, que foi desenvolvido utilizando a linguagem de programação Scala, é capaz de produzir soluções altamente escaláveis, tanto vertical quanto horizontalmente [Roestenburg et al. 2016]. Alem disso, o Akka fornece *routers* do tipo round-robin, random entre outros, para realizar o balanceamento de carga sobre um entorno local e distribuído.

4. Metodologia

A presente proposta para processar arquivos GRIB em um ambiente paralelo e distribuído é baseada no Akka toolkit, o qual implementa o Modelo de Atores. O Akka foi utilizado principalmente para a distribuição das tarefas através dos nós membros do cluster. Para realizar este processamento, foi desenvolvido um módulo capaz de parsear e decodificar os arquivos GRIB. Esse módulo, por sua vez, também foi escrito em Scala, para obtermos melhor compatibilidade com o Akka. A Figura 1 ilustra a arquitetura proposta, onde podemos distinguir 2 tipos de máquinas: o nó Master e o nó Worker. No primeiro reside o nó Seed (1), o qual é padrão do Akka e responsável por formar o cluster, servindo como ponto inicial de contato para que os demais nós conectem-se ao cluster. O outro refere-se a um nó Akka com o papel de Master (2), onde reside o ator JobManager. No segundo residem os nós Akka com o papel de Worker (3), que contém aos atores do tipo JobWorker.

O conjunto de dados analisado nos experimentos foi armazenado no HDFS (com um fator de replicação de 2). Por esse motivo, na figura encontram-se um Namenode no nó Master, e Datanodes nos outros nós que atuam como Workers.

O trabalho realizado pelo JobManager e pelo ator JobWorker é baseado no modelo de comunicação **Manager-Worker** [Chandy and Taylor 1992], onde o **Worker** realiza requisições ao **Manager** das tarefas a serem processadas. O funcionamento da nossa proposta é ilustrado na Figura 2(a). No *Driver program* são lidas as configurações necessárias para a execução da aplicação, obtém-se a lista de arquivos GRIB a serem processados do HDFS, e uma primeira messagem, JobRequest, é enviada ao ator JobManager. No JobManager, um *router* do tipo *Broadcast* é criado e posteriormente utilizado para enviar uma mensagem *InitProcess* aos atores JobWorker. Consequentemente, cada JobWorker envia

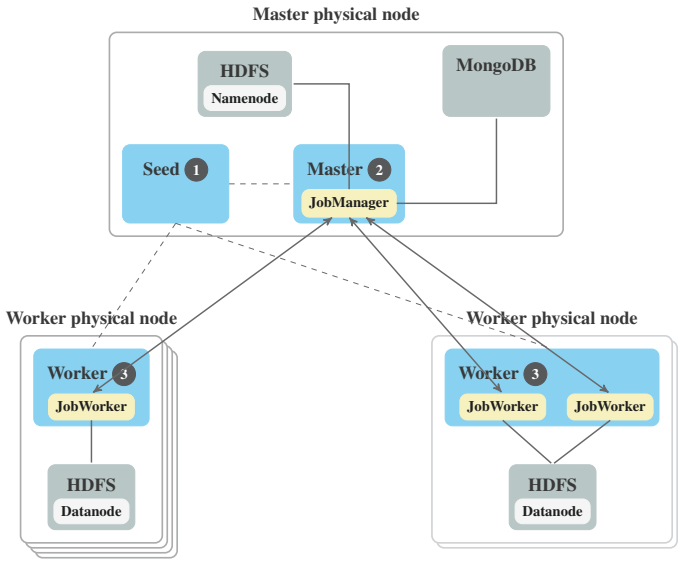


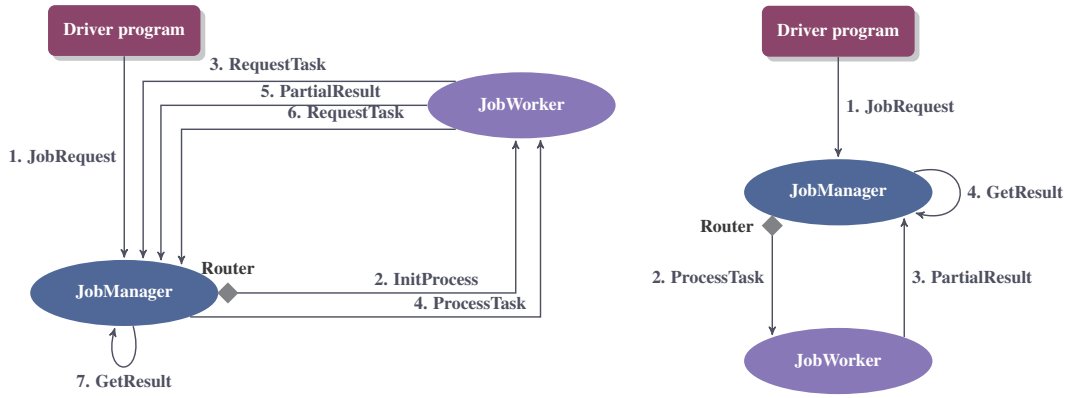
Figura 1. Arquitetura final proposta

uma mensagem *RequestTask* ao JobManager, para registrar-se como um executor de tarefas, e solicitar uma tarefa a ser processada. Todas as mensagens recebidas no JobManager são armazenadas na caixa de correio e atendidas na ordem de chegada. Desse modo, o JobManager seleciona uma tarefa e envia uma mensagem *ProcessTask* ao JobWorker com a informação da tarefa a ser realizada, porém dessa vez a mensagem é enviada diretamente, sem o uso do *router*. Nesse ponto, o ator JobWorker utiliza o módulo cliente em Scala para processar os arquivos GRIB armazenados no HDFS, e o resultado de cada mensagem GRIB é combinado para calcular um resultado parcial. Após calculado o resultado parcial, uma mensagem *PartialResult* é enviada ao ator JobManager, seguida de outra mensagem *RequestTask* solicitando uma nova tarefa a ser realizada. No JobManager os resultados parciais recebidos são persistidos em memória, e uma outra tarefa pendente é selecionada para ser enviada ao JobWorker através de uma mensagem *ProcessTask*. Esse ciclo se repete até que não hajam mais mensagens pendentes a serem processadas no JobManager e, quando isto acontece, o JobManager envia uma mensagem chamada *GetResult* a si mesmo, para calcular o resultado final a partir dos resultados parciais obtidos anteriormente.

Tabela 1. Número total de JobWorkers por cluster de VMs

Configuration	1 VM	2 VMs	4 VMs	8 VMs
4*WN* - 1*JW*	4	8	16	32
2*WN* - 2*JW*	4	8	16	32
4*WN* - 2*JW*	8	16	32	64
2*WN* - 4*JW*	8	16	32	64
4*WN* - 4*JW*	16	32	64	128
2*WN* - 8*JW*	16	32	64	128

A escolha das tarefas a serem processadas pelos atores JobWorker pode ocorrer de duas formas: Baseado em arquivo e baseado em mensagens. No primeiro, cada ator



(a) Usando o modelo Manager-Worker.

(b) Usando os routers fornecidos pelo Akka.

Figura 2. Fluxograma da nossa proposta e de uma segunda abordagem.

JobWorker processa um arquivo inteiro com todas as mensagens (175 por arquivo). Na escolha de tarefas baseada em mensagens, o JobManager particiona o arquivo em grupos de mensagens, podendo variar entre 22, 44 ou 88 mensagens, sendo essa etapa adicional realizada pelo JobManager. Esses números foram selecionados para dividir cada arquivo da forma mais equitativa possível. Dessa forma, se o número de mensagens a serem processadas por cada tarefa for 44, um arquivo GRIB será processado por quatro JobWorkers, onde 3 JobWorkers processarão 44 mensagens e 1 JobWorker processará apenas 43 mensagens.

Múltiplos atores JobWorker podem ser executados em cada nó Akka do tipo Worker. Através da Tabela 1, podemos visualizar o número de JobWorkers utilizado por cada conjunto de máquinas virtuais. Dessa forma, por exemplo, a configuração **2WN-4JW** mostra que 2 nós Akka do tipo Worker (**WN**) serão executado em cada máquina, dos quais cada um conterá 4 JobWorkers (**JW**). De modo que, para 1 VM serão criados 8 JobWorkers, para 2 VMs serão criados 16 JobWorkers, para 4 VMs serão 32 atores JobWorker e, por fim, para 8 VMs serão criados 64 JobWorkers. As demais configurações apresentadas na tabela seguem o mesmo padrão, conforme aumenta o número de máquinas virtuais.

Além disso, uma segunda abordagem é apresentada, com o objetivo de comparar a nossa proposta com relação a outras opções fornecidas pelo Akka. Os *routers* utilizados para realizar o estudo comparativo foram RoundRobin, Random e AdaptiveLoadBalancing. O AdaptiveLoadBalancing utiliza métricas do cluster para comunicar-se com os atores, podendo ser configurado para usar métricas tais como: *cpu*, para o nível de utilização da CPU; *heap*, que utiliza como métrica a pilha disponível da JVM; e a métrica *mix*, que combina as métricas de cpu, heap e load. Essas métricas foram coletadas utilizando a biblioteca Hyperic Sigar[9].

Figura 2(a) ilustra como essa outra abordagem funciona. De forma similar ao explicado anteriormente, os parâmetros necessários para a execução da aplicação são carregados no *Driver program* e uma primeira mensagem é enviada ao ator JobManager, denominada JobRequest, contendo a lista dos arquivos GRIB a serem processados. Nesse ator, um router será criado e o mesmo será encarregado de distribuir as mensagens aos

[9]https://github.com/hyperic/sigar

JobWorkers de acordo com a lógica do *router* especificado. O JobWorker, por sua vez, processará uma por uma as mensagens, *ProcessTask*, recebidas e armazenadas na caixa de correio do ator. O resultado parcial obtido será enviado ao JobManager na mensagem *PartialResult*. Esse processo continuará até não houverem mais mensagens a ser enviadas no JobManager e o JobWorkers tenham processado todas as tarefas recebidas. Quando isso acontecer, o JobManager enviará uma mensagem *GetResult* para si mesmo, e da mesma forma o resultado final será calculado a partir dos resultados parciais recebidos nas mensagens anteriores.

5. Resultados

Os experimentos foram executados usando Java 1.8.0 60, Scala 2.11.7, Akka 2.3.4 e Apache Hadoop 2.6.0, em um *cluster* no Windows Azure. Esta plataforma foi escolhida devido aos resultados em Roloff et al. [Roloff et al. 2012], onde o Windows Azure apresentou melhores resultados, com relação à eficiência de custo e desempenho, em comparação com os serviços em nuvem da Amazon e Rackspace. O cluster criado é composto por 9 instâncias do tipo A6, cada uma com 4 núcleos virtuais, 28 GB de memória e sistema operacional Ubuntu 14.04 LTS.

5.1. Descrição do conjunto de dados

O conjunto de dados usado nos experimentos contém 184 arquivos, cada um desses arquivos é composto de 175 mensagens GRIB, version 2 (GRIB2), em um conjunto total de 32200 mensagens, correspondentes a apenas três meses de dados. Esses arquivos foram obtidos através do FTP do CPTEC[10], onde encontram-se organizados em duas pastas por dia, uma correspondente às 00:00 e a outra às 12:00.

O tamanho total do conjunto de dados é de 13 GB, mas após decodificado e transformado em texto plano contendo todos os valores, o tamanho total é de 378 GB, o qual excede amplamente a quantidade de memória disponível no cluster utilizado. Este tamanho do conjunto de dados foi escolhido especificamente para acelerar a execução dos experimentos. Latitude-longitude é o tipo de grade utilizado para todas as mensagens GRIB. Além disso, cada arquivo é composto de 14 mensagens GRIB usando o **Complex packing** e 161 mensagens com o método de empacotamento **Complex packing and spatial differencing**. A maioria das mensagens contêm 1584353 pontos ou valores de dados. Da mesma forma, ao analisar todo o conjunto de dados foram encontrados 27 tipos diferentes de parâmetros. Assim, por exemplo, Temperatura, Umidade relativa e Precipitação Total tiveram no total 3864, 3496 e 184 ocorrências, respectivamente.

Para esse conjunto de dados, foram realizados experimentos abordando dois tipos de cenários. O primeiro cenário, realiza o cálculo do *average* para os parâmetros Temperatura e Umidade relativa. O segundo cenário, por sua vez, calcula o *average* para todos os parâmetros disponíveis no conjunto de dados.

Os resultados dos experimentos em 5.2 e 5.3 apresentam 6 métodos de execução. O método que utiliza a nossa proposta é denominado **pGrib**, enquanto os *routers round-robin* e *random* são denotados como **rr** e **ra**, respectivamente. Finalmente, o router *adaptive load balancing*, utilizando as métricas de CPU, Heap e mix, são denotados respectivamente como **cmc**, **cmh** e **cmm**. Assim, para cada um destes métodos, foram executados

[10]ftp://ftp1.cptec.inpe.br/modelos/io/tempo/regional/BRAMS05km/grib/

um total de 24 combinações, sendo elas: **4WN-1JW**, **2WN-2JW**, **4WN-2JW**, **2WN-4JW**, **4WN-4JW**, e **2WN-8JW**. Para cada um das configurações descritas, o tamanho do numero de mensagens processadas por cada JobWorker varia entre **22**, **44**, **88** e **175**.

5.2. Primeiro cenário

Neste cenário, são apenas considerados os parâmetros Temperatura e Umidade relativa. A Figura 3 apresenta os resultados para cada um dos métodos de execução, considerando o melhor e o pior caso obtidos. Para este cenário, os melhores casos foram obtidos usado 2WN e variando o número de JW por cada WN entre 4 e 8 na maioria dos casos.

Neste mesmo cenário, para a nossa proposta, baseando-se em arquivo ou grupo de mensagens, os melhores casos foram obtidos com a configuração 2WN-4JW. A única exceção ocorreu quando utilizamos 8 VMs e as mensagens foram agrupadas. Nesse caso, a melhor configuração foi 2WN-2JW-88, que ocupou 30.89 segundos a mais do que a 2WN-4JW-175. Dessa forma, houve uma ligeira variação de 1.6% para 1 VM, 2.55% para 2 VMs, 1.54% para 4 VMs, e 5.21% para 8 VMs entre a melhor e segunda melhor combinação para a nossa proposta.

A segunda melhor opção para processar os arquivos GRIB foi obtida utilizando-se o método rr. Os piores casos, por outro lado, foram obtidos utilizando 4WN e 1JW. O rr mostrou uma diferença significativa em relação aos outros métodos.

Quando utilizaram-se 8 VMs como *workers* o pGrib obteve um ganho significativo de 35.21%, 42.72%, 43.2%, 42.28% e 42.74% em relação aos métodos rr, ra, cmc, cmh e cmc, respectivamente.

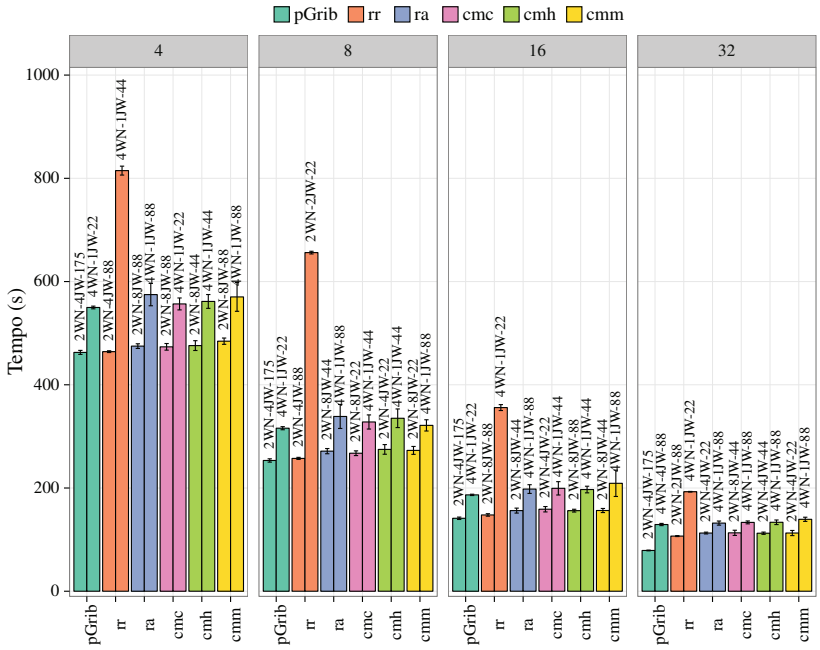


Figura 3. Tempo de execução do melhor e pior caso de cada um dos métodos de execução.

O melhor e pior speedup alcançado por todos os métodos são mostrados na Figura 4. O *baseline* utilizado foi o tempo de execução sequencial do módulo cliente, implementado para parsear os arquivos GRIB.

A Figura 4 mostra que, com o aumento do número de nós, todos os métodos obtêm boa escalabilidade, especialmente o método proposto neste trabalho. No entanto, podemos observar que para o pGrib o speedup diminui ligeiramente após 4 VMs, enquanto para os outros métodos o speedup diminui consideravelmente. Uma possível explicação para isto é que nem todas as mensagens são processadas pelo JobWorker.

Por exemplo, considerando o caso em que 1 JobWorker precise processar 44 mensagens, para as quais apenas 16 mensagens são Temperatura e Umidade relativa, 28 mensagens não seriam mais necessárias. Ainda assim, algumas seções da mensagem precisam ser analisadas, para sabermos se a mensagem GRIB precisa ser decodificada ou não. Esse processamento, por sua vez, requer leituras adicionais de dados no HDFS e processamento parcial das mensagens, acarretando as custos adicionais de processamento.

No entanto, ao implementar o modelo Manager-Worker, a nossa proposta consegue um balanceamento de carga automático, resultando em uma melhor distribuição das tarefas.

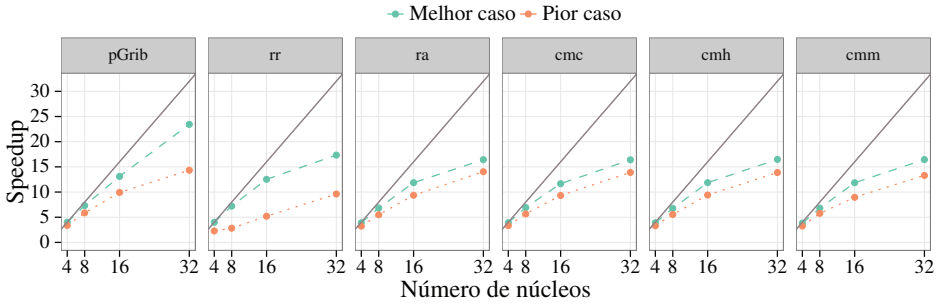


Figura 4. Speedup do melhor e pior caso de cada um dos métodos de execução.

5.3. Segundo cenário

Neste cenário, são considerados todos os parâmetros do conjunto de dados. A Figura 5 apresenta o tempo gasto por cada um dos métodos de execução, levando em conta o melhor e o pior caso. Considerando apenas uma VM, o melhor tempo foi obtido usando o rr, com um ganho de 1% em relação ao pGrib. Ao utilizarmos 2 VMs, 4 VMs e 8 VMs, o pGrib obteve uma melhor performance ao processar os arquivos GRIB quando dividimos os arquivos em grupos de 88, 44 e 22 mensagens, respectivamente. De forma semelhante ao experimento anterior, os melhores resultados para a nossa proposta foram obtidos utilizando-se a configuração 2WN-4JW. Da mesma forma, os piores resultados foram usando 4WN e apenas 1JW por cada WN.

Ao utilizarmos 8 VMs como *workers*, o pGrib obteve um ganho de 3.22%, 11.36%, 11.78%, 15.09% e 11.7%, respectivamente, em relação aos métodos rr, ra, cmc, cmh e cmc.

O speedup dos melhores e piores casos para este segundo cenário é apresentado na Figura 6, onde podemos perceber diferenças em relação ao cenário anterior. Os melhores valores de escalabilidade, para todos os métodos de execução, foram obtidos utilizando-se todos os parâmetros. Isso decorre do fato de não ocorrerem leituras no HDFS, nem seções das mensagens GRIB sendo analisadas desnecessariamente, resultando assim em um melhor desempenho.

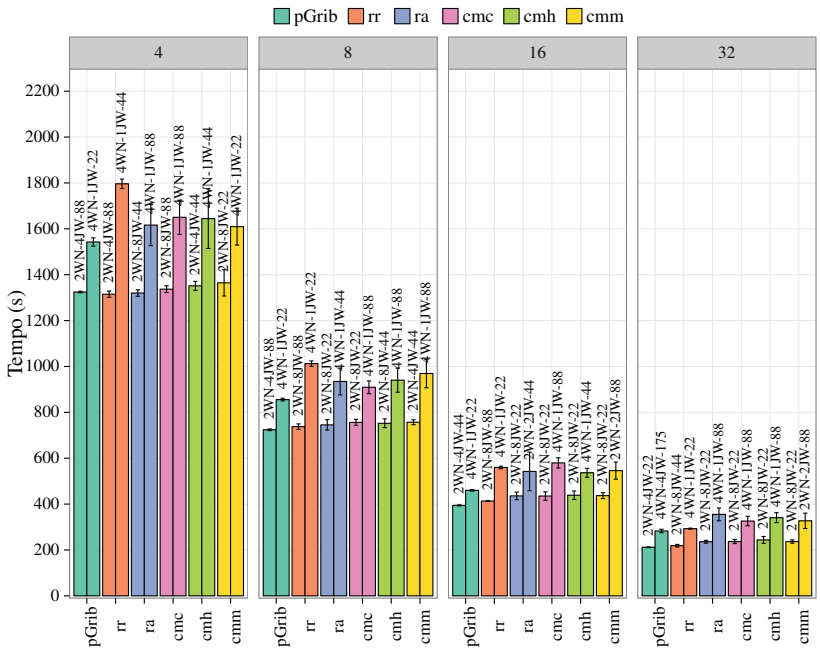


Figura 5. Tempo de execução do melhor e pior caso de cada um dos métodos de execução.

5.4. Usando Metadados

Após analisar os resultados do primeiro cenário, foi observada uma diferença entre a abordagens baseada em arquivo e por agrupamento por mensagens, o que resultou em uma perda de 28.11% na segunda abordagem, dentre as melhores configurações de cada estratégia. Esta perda é resultante do fato de que, na abordagem baseada em agrupamento de mensagens, quando precisamos escolher uma tarefa a ser enviada, o JobManager precisa realizar um pré-processamento, a fim de identificar o começo de cada mensagem GRIB dentro do arquivo, e posteriormente enviar cada grupo de mensagens ao JobWorker para ser processado. Porém, todo o pré-processamento é realizado sob demanda no JobManager, por arquivo, resultando em uma perda de desempenho. No segundo cenário, no entanto, essa diferença não ocorre. Devido ao fato de que todos os parâmetros são processados, não ocorrendo leituras desnecessárias no HDFS.

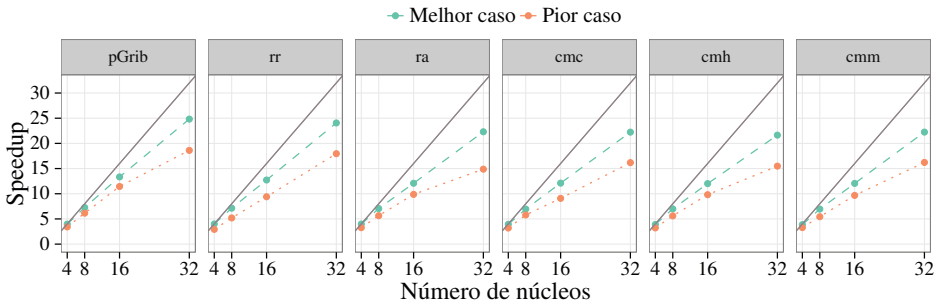


Figura 6. Speedup do melhor e pior caso de cada um dos métodos de execução.

Por esse motivo, foi realizada uma otimização sobre a abordagem baseada no agrupamento de mensagens. A otimização consiste em realizar o pré-processamento dos arquivos GRIB, para obter a ponto de início e o parâmetro que identifica cada mensagem GRIB (podendo ser salvas outras informações). O resultado desse processamento foi

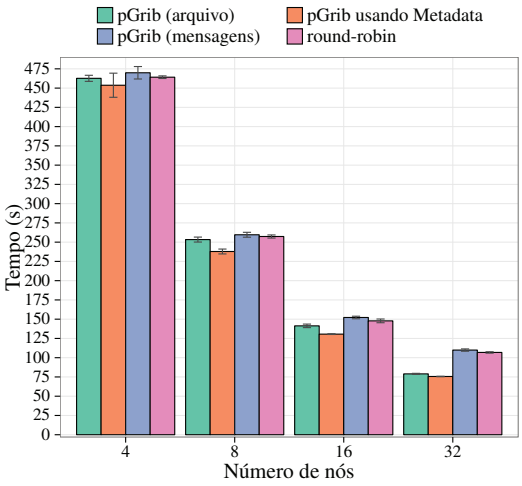


Figura 7. Tempo de execução para os melhores caso obtidos no primeiro cenário.

armazenado no banco de dados MongoDB, de forma que o JobManager possa realizar uma consulta ao MongoDB para obter os metadados relacionados a um arquivo. O pré-processamento e o posterior armazenamento em MongoDB resultou em um tempo total de 179.47 segundos, porém é uma operação realizada uma única vez, podendo ser reutilizada em todas as demais execuções.

A Figura 7 mostra os tempos de execução do pGrib com a estratégia baseada em arquivo, em mensagens, usando metadados, e round-robin. Como esperado, o uso de metadados obteve resultados satisfatórios. De forma que, levando em consideração a utilização de apenas uma VM, a otimização resultou em um ganho de 1.98% em relação ao pGrib (baseado em arquivo). Ao utilizarmos 2 e 4 VMs, a amplitude do ganho foi ainda maior, resultando em ganhos de 6.51% e 8.23% respectivamente. Ao utilizarmos 8 VMs, o ganho foi de 4.48%. Porém, com relação ao pGrib (baseado em mensagens) e round-robin, o ganho foi de 45.34% e 41.27%, respectivamente. Embora não exista um grande ganho entre os melhores casos da proposta inicial e a otimização realizada, existe um ganho considerável em relação aos resultados obtidos com a estratégia baseada em mensagem, o qual era o objetivo desta otimização.

5.5. Comparação com Apache Spark

Neste experimento, apresentamos uma comparação entre a nossa proposta e uma versão que processa os arquivos GRIB utilizando o Apache Spark (versão 1.3.1). Para este caso, a primeira abordagem contemplada foi a utilização do módulo cliente Scala para analisar e decodificar os arquivos GRIB. Porém, ao persistirmos os dados decodificados nos RDDs, ocorriam exceções do tipo *OutOfMemory*, já que os valores de dados decodificados ocupam mais memória do que o disponível. Dessa forma, optamos por armazenar nos RDDs apenas os resultados obtidos do módulo cliente Scala, da mesma forma como é realizado em nossa proposta. Assim, quando utilizada 1 VM, houve 1 *executor* e 4 *cores*, para 2, 4 e 8 VMs usou-se 2, 4 e 8 *executors* e 4 *cores* por *executor*, além de 24GB por *executor*.

A Figura 8 mostra os tempos de execução para o primeiro cenário (lado esquerdo), e para o segundo cenário (lado direito), utilizando os melhores casos obtidos com a nossa

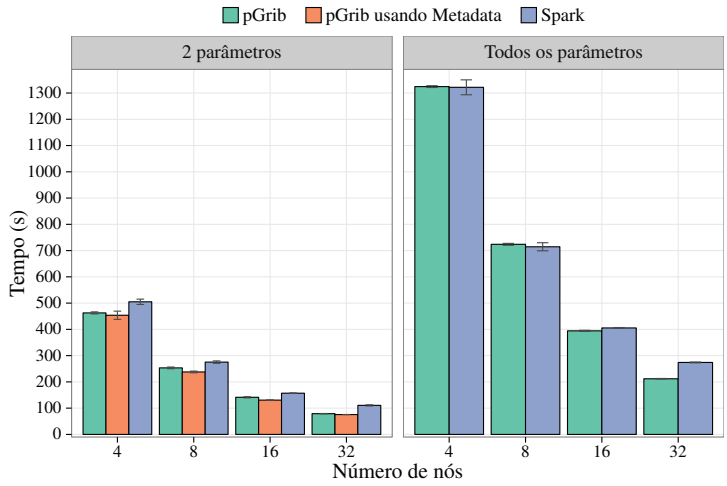


Figura 8. Tempo de execução para o melhor caso com a nossa proposta e o Apache Spark.

proposta e com o Apache Spark. De acordo com o primeiro cenário, o pGrib com o uso de metadados obteve um ganho de 11.3%, 15.74%, 20.19% e 46.31% usando 1, 2, 4 e 8 VMs, respectivamente. Para o segundo cenário, utilizando 1 e 2 VMs, o Spark foi ligeiramente melhor com um ganho de 0.21% e 1.32%, respectivamente, em relação ao pGrib com processamento baseado em mensagens. No entanto, ao utilizarmos 4 VMs, o pGrib obteve um ganho de 2.8%. Enquanto com 8 VMs o ganho foi de 29.35%. Uma explicação para a obtenção da performance superior ao utilizarmos 8 VMs, deve-se ao fato que inicialmente os arquivos GRIB foram armazenados no HDFS utilizando apenas 4 nós. De modo que, para 8 nós, uma maior transmissão de dados é necessária, causando uma perda de desempenho no Apache Spark, que em seu comportamento padrão precisa ter todos os dados disponíveis na memória antes de realizar as operações.

6. Trabalhos Relacionados

Existe um grande esforço de pesquisa sobre o processamento de grandes quantidades de dados científicos em ambientes paralelos e distribuídos. Porém, estes trabalhos são focados principalmente nos formatos NetCDF [Rew and Davis 1990] e HDF5 [The HFD Group 2016]. Li et al. [Li et al. 2003] propõem uma interface paralela para ler e escrever arquivos NetCDF, o qual está baseado em MPI-IO. Zhao et al. [Zhao et al. 2010] propõem um método para armazenar e acessar volumes massivos de dados de dados no formato NetCDF, baseado em processamento através do Apache Hadoop. No entanto, na fase inicial, os conjuntos de dados precisam ser analisados e transformados em formato texto para serem armazenados no HDFS, causando assim uma sobrecarga no volume de armazenamento de dados desnecessário.

Buck et al. [Buck et al. 2011], implementaram o SciHadoop, um *plugin* sobre o Apache Hadoop que permite o processamento de dados, em formato NetCDF, assim como a execução de consultas lógicas sobre esse modelo. Nesse trabalho, aplicam a função *mediana* sobre os dados para um determinado intervalo de tempo, e um determinado intervalo de área para analisar a pressão de ar. Além disso, neste trabalho, estendem a biblioteca NetCDF-Java para trabalhar com HDFS. O SciMate Framework, apresentado

por Wang et al. [Wang et al. 2012], é uma ferramenta que permite processar os formatos de dados científicos NetCDF e HDF5, assim como arquivos de texto puros, permitindo também a extensibilidade para outros formatos de dados científicos. Esse trabalho é baseado no sistema Mate, o qual permite processar dados usando o modelo de programação MapReduce. Blanas et al. [Blanas et al. 2014] propuseram um sistema que pode realizar a análise de dados científicos diretamente sobre conjuntos de dados armazenados no formato HDF5, executando as consultas em memória e usando a indexação através de *bitmap*, aproveitando-se das grandes quantidades de memória disponíveis nos supercomputadores.

7. Conclusões

Este trabalho apresenta uma análise de desempenho do processamento de arquivos GRIB, utilizando o modelo de comunicação Manager-Worker implementado no modelo de atores. Foi realizada uma comparação entre a nossa proposta, os *routers* suportados pelo Akka tookit que fornecem escalonadores de tarefas, e o Apache Spark. Além disso, foi realizada uma otimização sobre a proposta inicial, com o objetivo de evitar leituras do HDFS e decodificação desnecessária das mensagens GRIB. Os resultados obtidos demostraram a escalabilidade da solução, quando variados o número de máquinas virtuais e o número de atores por nó do Akka. Dado que os atores não são mapeados em uma relação 1-1 com as *threads* do sistema, e que atores são uma primitiva de alto nível implementada com base em *threads* JVM gerenciadas pelo Akka, termos apenas um ator por nó do Akka não apresentou uma boa eficiência. Por outro lado, dispararmos mais atores por nó do Akka resultou em uma melhor performance.

Como trabalho futuro, planejamos realizar uma análise sobre o comportamento dos atores JobManager e JobWorker para conseguir um melhor paralelismo, generalizar a implementação proposta para ser capaz de processar outros formatos de dados científicos.

8. Agradecimentos

Esta pesquisa recebeu financiamento do Programa H2020 da União Européia e do MCTI/RNP-Brasil no âmbito do projeto HPC4E, contrato de subvenção No. 689772. Esta pesquisa também recebeu financiamento da FAPERGS, no âmbito do projeto GreenCloud.

Referências

Blanas, S., Wu, K., Byna, S., Dong, B., and Shoshani, A. (2014). Parallel Data Analysis Directly on Scientific File Formats. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14.

Buck, J. B., Watkins, N., LeFevre, J., Ioannidou, K., Maltzahn, C., Polyzotis, N., and Brandt, S. (2011). SciHadoop: Array-based Query Processing in Hadoop. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11.

Candanedo, J. A., Paradis, E., and Stylianou, M. (2013). Building Simulation Weather Forecast Files for Predictive Control Strategies. In *Proceedings of the Symposium on Simulation for Architecture & Urban Design*, SimAUD '13.

Chandy, K. and Taylor, S. (1992). *An introduction to Parallel Programming*. Jones and Bartlett.

Dobre, C. and Xhafa, F. (2014). Parallel Programming Paradigms and Frameworks in Big Data Era. *International Journal of Parallel Programming*, 42(5).

Fortner, B. (1995). *The Data Handbook: A Guide to Understanding the Organization and Visualization of Technical Data*. Springer-Verlag.

Gu, L. and Li, H. (2013). Memory or Time: Performance Evaluation for Iterative Operation on Hadoop and Spark. In *High Performance Computing and Communications 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on*.

Hewitt, C., Bishop, P., and Steiger, R. (1973). A Universal Modular ACTOR Formalism for Artificial Intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73.

Kambatla, K., Kollias, G., Kumar, V., and Grama, A. (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing* , 74(7):2561 – 2573. Special Issue on Perspectives on Parallel and Distributed Processing.

Li, J., Liao, W.-k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003). Parallel netCDF: A High-Performance Scientific I/O Interface. In *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03.

Markiewicz, Ł., Chybicki, A., Drypczewski, K., Bruniecki, K., and Dbrowski, J. (2013). Operational Enhancement of Numerical Weather Prediction with Data from Real-time Satellite Images. In Weintrit, A., editor, *Marine Navigation and Safety of Sea Transportation: Navigational Problems*, chapter 5, pages 153–160.

Rew, R. and Davis, G. (1990). NetCDF: an interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82.

Roestenburg, R., Bakker, R., and Williams, R. (2016). *Akka in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition.

Roloff, E., Birck, F., Diener, M., Carissimi, A., and Navaux, P. (2012). Evaluating High Performance Computing on the Windows Azure Platform. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*.

The HFD Group (1987-2016). Hierarchical Data Format, version 5.

Wang, Y., Jiang, W., and Agrawal, G. (2012). SciMATE: A Novel MapReduce-Like Framework for Multiple Scientific Data Formats. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12.

WMO (2003). Guide to the WMO Table Driven Code Form Used for the Representation and Exchange of Regularly Spaced Data In Binary Form: FM 92 GRIB Edition 2. Technical report, World Meteorological Organization.

Zhao, H., Ai, S., Lv, Z., and Li, B. (2010). Parallel Accessing Massive NetCDF Data Based on Mapreduce. In *Proceedings of the 2010 International Conference on Web Information Systems and Mining*, WISM'10.