Spark-SGreedy: Um Algoritmo de Escalonamento de *Workflows* Intensivos em Dados no *Framework* Apache Spark

Victor F. de Sousa[1], **Cristina Boeres**[1], **Daniel de Oliveira**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (IC/UFF)

{boeres,danielcmo,victorsousa}@ic.uff.br

Resumo. *Nos últimos anos, o Apache Spark vem sendo utilizado como framework para execução de experimentos científicos modelados como workflows. Por mais que represente um avanço, o Spark não foi projetado para gerenciar execuções de aplicações científicas, e seu escalonamento não considera estimativas de consumo de recursos pelas atividades do workflow. Esse artigo apresenta o Spark-SGreedy, uma proposta de algoritmo de escalonamento de workflows no Spark que usa dados de proveniência (histórico) para analisar a previsão de consumo de recursos das atividades do workflow e escaloná-las de acordo com tal previsão.*

1. Introdução

O uso de simulações computacionais de larga escala para executar experimentos científicos ganhou importância nos últimos anos [de Oliveira et al. 2019]. Tais experimentos são compostos pelo encadeamento de programas que realizam operações sobre os dados cientıficos a serem analisados, tais como carga (*data loading*) e processamento (*data processing*). Uma abstração utilizada para modelar esses experimentos são os *workflows*. Um *workflow* é representado por meio de um DAG, onde os vértices representam as atividades (programas) que executam operações sobre os dados, enquanto as arestas representam as dependências entre elas [de Oliveira et al. 2019]. Muitos *workflows* são de larga escala, e podem executar por horas ou dias, dependendo do volume de dados de entrada. Dessa forma, requerem execuções em ambientes de Processamento de Alto Desempenho (PAD) aliados à aplicação de técnicas de paralelismo. Eles são comumente executados em Sistemas de Gerência de *workflows* (SGWfCs). Entretanto, os SGWfs possuem uma curva de aprendizado alta e nem todo *workflow* pode ser modelado nos SGWfs de forma simples.

Uma opção interessante aos SGWfs para a execução de *workflows* é o uso de Ambientes de Computação Escalável e Intensiva em Dados (*i.e.*, DISC) e seus *frameworks*. Diferentemente dos ambientes PAD (que são centrados em computação), os ambientes DISC [Bryant 2011], são centrados em dados e concentram-se na tolerância à falhas e escalabilidade usando *clusters* de máquinas com *hardware* comum (*i.e.*, *commodity*). Os ambientes DISC se baseiam em operadores de processamento de dados (Map, Reduce, etc). Muitos *frameworks* DISC se encontram disponíveis para uso, como o Apache Spark [Zaharia et al. 2016], relevantes atualmente uma vez que permite o processamento de dados em memória. Apesar de *frameworks* DISC representarem um enorme avanço no que tange o apoio a experimentação cientıfica, ainda existem diversas barreiras para seu uso. Os *frameworks* DISC não foram originalmente projetados para gerenciar *workflows*, que possuem requisitos diferentes de aplicações comerciais no que tange a execução, e principalmente o escalonamento de atividades. O escalonamento padrão do Spark não considera estimativas de consumo de memória e CPU das atividades. Além disso, algoritmos tradicionais de escalonamento como o HEFT [Topcuoglu et al. 2002] e o SGreedy [de Oliveira et al. 2012] não podem ser portados diretamente para o Spark, pois não consideram dados em memória nem as características de ambientes DISC.

Assim, o objetivo do trabalho apresentado nesse artigo é desenvolver um algoritmo de escalonamento de *workflows* no *framework* Spark chamado Spark-SGreedy, de modo a apoiar transversalmente as diferentes disciplinas científicas que migram seu processo científico para projetos baseados no Spark. Exemplos de *workflows* que podem se beneficiar deste tipo de algoritmos de escalonamento são os *workflows* para geração de árvores filogenéticas. A Seção 2 discute sucintamente a abordagem proposta e a Seção 3 apresenta as conclusões.

2. Abordagem Proposta: O Spark-SGreedy

Para discutirmos um algoritmo de escalonamento de *workflows* no Spark, devemos primeiramente definir formalmente um *workflow*. Um *workflow* W pode ser definido por meio de um DAG, onde os nós $A = \{a_1,a_2, \ldots, a_n\}$ representam as atividades e as arestas Dep representam a dependência de atividades entre a atividade em A. Dado $a_i \mid (1 \leq i \leq n)$, e seja $I = \{i_1, i_2, .., i_m\}$ os dados de entrada para uma atividade a_i, então $input(a_i) \subset I$. Além disso, consideremos O como o conjunto de dados de saída produzidos por uma atividade a_i, então $output(a_i) \subset O$. As dependências entre duas atividades a_i e a_j é representada por $dep(a_i, a_j) \leftrightarrow \exists O_k \in input(a_j) | O_k \in output(a_i)$. De forma a escalonar atividades de *workflows* no Spark considerando consumo de recursos de cada atividade do *workflow*, devemos possuir dados históricos que nos ajudem a estimar o consumo de cada atividade para realizarmos o escalonamento.

Recentemente [Guedes et al. 2020] propuseram uma extensão do Spark chamada SAMbA. O SAMbA coleta dados históricos da execução de aplicações Spark, chamados dados de proveniência [Freire et al. 2008]. A partir da proveniência, é possível estimar o tempo de execução de atividades considerando os mecanismos em memória do Spark, como os RDDs e *Data Frames*. Todos os dados coletados pelo SAMbA estão disponíveis para consulta em um banco de dados. Assim, a proposta de algoritmo de escalonamento para o Spark implementada nesse artigo é uma extensão do algoritmo SGreedy [de Oliveira et al. 2012], chamada de Spark-SGreedy, que é apresentado no Algoritmo 1. O Spark-SGreedy recebe como entrada, o *workflow* a ser executado, um prazo limite, um orçamento limite e pesos para serem aplicados nos critérios de tempo de execução e custo financeiro. O algoritmo é guloso e sempre que uma máquina se encontra ociosa, uma nova atividade é enviada para o seu processamento. A escolha da melhor atividade para a máquina ociosa se dá por meio do cálculo do tempo de execução $T_n(a_i, vm_j)$ e seu custo financeiro $F_n(a_i, vm_j)$. É importante ressaltar que $T_n(a_i, vm_j)$ é calculado de acordo com a estrutura usada no Spark, seja ela RDD ou *Data Frame*. Todas as possíveis atividades são avaliadas e a que apresentar o valor mínimo para esse cálculo do custo é a escolhida. Esse processo se repete sempre que uma máquina ficar ociosa ou até que todas as atividades do *workflow* sejam executadas.

O desenvolvimento do Spark-SGReedy é um trabalho em andamento, mas experimentos preliminares com a execução do *workflow* Montage (`http://montage.ipac.caltech.edu/`) mostraram que a demanda por recursos e o tempo de execução de suas atividades são bastante heterogêneos, o que justifica um algoritmo de escalonamento como o Spark-SGreedy. Por exemplo, a atividade *mProjectPP* possui tempo de execução médio 13,99 min (desvio padrão de 2,12), enquanto que a atividade *mBackground* possui tempo execução médio de 10,64 min (desvio padrão 1,25), mas com um consumo de memória 125% maior que a atividade *mProjectPP*. Experimentos futuros utilizarão o Montage como *benchmark*.

3. Conclusões

Os *frameworks* DISC como o Apache Spark vêm sendo utilizados para execução de experimentos científicos modelados como *workflows*. Apesar de representar um avanço, os algorit-

mos de escalonamento do Spark não consideram características importantes de atividades de *workflows* científicos como consumo de CPU e memória. Esse artigo apresentou a proposta do Spark-SGReedy, um algoritmo de escalonamento de atividades de *workflows* que usa dados de proveniência (histórico) como base de forma a considerar requisitos das atividades no processo de escalonamento. O Spark-SGreedy é um trabalho em andamento, mas avaliações parciais mostraram que a heterogeneidade de consumo de recursos de *workflows* justifica o seu desenvolvimento.

Algorithm 1: Spark-SGreedy

W:o *workflow* para ser executado; E_{VM}:conjunto de máquinas; $Prazo$: prazo de execução;
$Orcamento$: orçamento limite; ω_t:peso para tempo de execução; ω_m:peso para custo financeiro; **Saída**: um escalonamento φ para W

$\varphi(W, E_{VM}) \leftarrow \emptyset$
$Disponivel \leftarrow \{a_i \in A | \forall a_j \in A \wedge \neg dep(a_i, a_j)\}$
$Prontas \leftarrow \{vm_i \in VM | \forall vm_j \in VM \wedge Ociosa(vm_j)\}$
while $Disponivel \neq \emptyset$ **do**
  for $vm_j \in Prontas$ **do**
    for $a_i \in Disponivel$ **do**
      $Custo \leftarrow \omega_t * T_n(a_i, vm_j) + \omega_m * F_n(a_i, vm_j)$
      $Possivel \leftarrow Possível + \{sched(a_i, vm_j, Custo)\}$
    end
    $Escolhido \leftarrow \min(Possível)$
    $\varphi(W, E_{VM}) \leftarrow \varphi(W, E_{VM}) + Escolhido$
    $Executa(Escohido)$
    $Pronto \leftarrow ready - \{vm\} + \{vmj \in VM | Ociosa(vm_j)\}$
    $Disponivel \leftarrow \{a_i \in A | a_j \in AC \wedge \neq dep(a_i, a_j)\}$
  end
end
Retorna $\varphi(W, E_{VM})$

Referências

Bryant, R. E. (2011). Data-intensive scalable computing for scientific applications. *Computing in Science Engineering*, 13(06):25–33.

de Oliveira, D., Ocaña, K. A. C. S., Baião, F. A., and Mattoso, M. (2012). A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.*, 10(3):521–552.

de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Comput. Sci. Eng.*, 10(3):11–21.

Guedes, T., Martins, L. B., Falci, M. L. F., Silva, V., Ocaña, K. A. C. S., Mattoso, M., Bedo, M. V. N., and de Oliveira, D. (2020). Capturing and analyzing provenance from spark-based scientific workflows with samba-rap. *Future Gener. Comput. Syst.*, 112:658–669.

Topcuoglu, H., Hariri, S., and Wu, M. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distributed Syst.*, 13(3):260–274.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65.