Uma Proposta de Implementação de Álgebra de Workflows em Apache Spark no Apoio a Processos de Análise de Dados

João Ferreira[1], Daniel Gaspar[2], Bernardo Monteiro[1], Ana Beatriz Cruz[1], Fabio Porto[2], Eduardo Ogasawara[1]

[1] CEFET/RJ

[2]LNCC - DEXL Lab

joao.parana@acm.org, gaspar@lncc.br

anacruz@acm.org, fporto@lncc.br, eogasawara@ieee.org

Abstract. *The typical activity of a Data Scientist involves the implementation of various processes that characterize data analysis experiments. In these analyzes there is a need to execute several codes in different programming languages (Python, R, C, Java, and Scala) in different parallel and distributed processing environments. Depending on the complexity of the process and the numerous possibilities for distributed execution of these solutions, it may be necessary to spend a lot of energy on different implementations that take the Data Scientist away from his ultimate goal of producing knowledge from large volumes of data. In this context, this paper aims to support this difficulty by proposing the construction of a framework conceived from an algebraic approach that isolates the process modeling from the difficulty of optimally executing such workflows. The proposal presents a typical Extract Transform Load (ETL) workflow for performing data analysis and points to promising results in terms of representation and abstraction/isolation potential of the execution environment.*

Resumo. *A atividade típica de um Cientista de Dados envolve a implementação de diversos processos que caracterizam experimentos de análise de dados. Nestas análises há a necessidade de executar diversos códigos em diferentes linguagens de programação (Python, R, C, Java e Scala) em diferentes ambientes de processamento paralelo e distribuído. Dependendo da complexidade do processo e das inúmeras possibilidades para execução distribuída destas soluções, pode ser necessário gastar muito energia em diferentes implementações que afastam o Cientista de Dados do seu objetivo final, que é produzir conhecimento a partir dos grandes volumes de dados. Dentro deste contexto, este trabalho visa apoiar na solução de tal dificuldade ao propor a construção de um framework concebido a partir de uma abordagem algébrica que isola a modelagem do processo da dificuldade de executar, de modo otimizado, tais workflows. A proposta apresenta um workflow típico de Extract Transform Load (ETL) para realização de análise de dados e aponta para resultados promissores em termos de representação e potencial de abstração/isolamento do ambiente de execução.*

1. Introdução

A atividade típica de um Cientista de Dados envolve a implementação de diversos processos que caracterizam experimentos de análise de dados. Tais experimentos dão origem a

workflows que possuem características empíricas e necessitam garantir reprodutibilidade. Em diversas situações, usar sistemas de gerência de workflows (SGW) é uma escolha natural para apoiar experimentos datacêntricos. Muitos trabalhos e implementações de SGW têm sido propostos e avaliadas [Liu et al., 2015].

Em contrapartida, no cenário de processamento de alto desempenho para *Big Data*, têm-se soluções baseadas no ecossistema *Spark*/*Hadoop* capazes de processar alto volume de dados em ambientes distribuídos [Salloum et al., 2016]. Este ecossistema de software, no entanto, impõe algumas limitações: *i)* a programação em linguagem procedural imperativa; *ii)* a falta de um mecanismo de gestão de proveniência sobrecarrega o cientista de dados com a necessidade de resolver problemas relacionados a infraestrutura de hardware e software para garantir uma correta coleta de dados da proveniência; *iii)* ausência de primitivas para apoiar cenários típicos de análise de dados como varredura de parâmetros, por exemplo. Em outras palavras, tais soluções compreendem um conjunto de APIs que viabiliza aos usuários na execução de tarefas paralelizáveis de forma distribuída, mas que não cobrem as funcionalidades gerais de SGWs.

Ademais, tem-se a disposição milhares de pacotes de softwares científicos e de engenharia desenvolvidos ao longo de anos de pesquisa tais como CRAN [2017] que possui mais de dez mil pacotes de software escritos em R e bibliotecas como SciPy [2017] em Python que disponibilizam centenas de pacotes completos para cientistas de todas as áreas de conhecimento. Tais bibliotecas são comumente usadas diretamente pelos Cientistas de Dados, estabelecendo cenários típicos de códigos legados a serem invocados nos workflows.

A proposta desse trabalho combina as características presentes na linguagem Scala com as funcionalidades do ecossistema *Spark*/*Hadoop* para desenvolver um *framework*, usando inversão de controle [Fayad and Schmidt, 1997], capaz de prover uma solução semelhante aos SGW, codificada de modo simples e, ao mesmo tempo, capaz de executar de modo eficiente. Em participar, a otimização das execuções de atividades em ambientes distribuídos que respeite, ao mesmo tempo, as características intrínsecas desses workflows com a grande presença de aplicações consoles e de bibliotecas e códigos legados escritos em Python e R.

Além desta introdução, o trabalho está organizado em mais cinco seções. Na seção 2, são apresentados conceitos gerais de workflow e as iniciativas de pesquisa nesta área. Em seguida, na seção 3, apresentam-se as características mais relevantes do *Spark* e seu ecossistema. Estes conceitos servem de base para o projeto do framework usando álgebra de workflows [Ogasawara et al., 2011], denominado de *Workflow Framework*, descritos na seção 4. A seção 5 discute a implementação da proposta por meio da descrição de um workflow de *ETL* (do inglês, *Extract Transform Load*) para aplicações em mobilidade urbana. Finalmente, a seção 6 apresenta as considerações finais.

2. Workflows Datacêntricos

Um workflow geralmente se refere ao processo de execução tanto de tarefas automatizadas, quanto manuais (com interação humana), integradas num mesmo fluxo, o qual tem por objetivo a produção de um determinado resultado [Van Der Aalst et al., 2003]. Em função da complexidade de tal execução, torna-se necessário gerenciá-la por meio de SGW. Os SGWs necessitam de linguagens de especificação de workflows, as quais

podem ser: (i) gráficas (normalmente associadas a grafos ou redes de Petri); (ii) baseadas em XML (XPDL etc); ou (iii) baseadas linguagens de especificação próprias *plain-text* [Deelman et al., 2009]. Em uma visão mais datacêntrica, um workflow é composto de atividades onde cada uma delas é um componente de software capaz de executar programas considerando parâmetros de entrada e de saída [Ogasawara et al., 2011]. Tais parâmetros são usados para definir as dependências entre as atividades de um workflow.

Diversos cenários de análise de dados podem ser modelados como grafos acíclicos direcionados (DAG). Em particular, também, são muito comuns problemas que recaem em varredura de parâmetros, *i.e.*, as análises devem ser executadas com vários conjuntos de parâmetros diferentes para que se possa escolher o melhor resultado. Isso vem da característica empírica dos experimentos científicos *in-silico*. Por exemplo, num experimento de mineração de dados, para treinar modelos de classificação é necessário executar vários algorítimos diferentes com vários parâmetros diferentes para encontrar o modelo que apresenta melhor comportamento para um dado problema específico. Desta forma, mecanismos que otimizem a produção e consumo de resultados intermediários e otimize quais atividades devem ser executadas antes e quais delas devem ser executadas de modo agrupado (*pipeline*) passam a ser muito relevantes. As abordagens algébricas [Jergler et al., 2015; Rheinländer et al., 2015; Ogasawara et al., 2011] procuram otimizar a execução dos workflows, levando-se em consideração o contexto de execução, os dados com os seus respectivos metadados e as informações de proveniência.

3. Spark e o Ecossistema Hadoop

Salloum et al. [2016] descrevem os cenários atuais de BigData e as funcionalidades mais relevantes do *Spark* que proveem desempenho no processamento e gerência de dados no ecossistema *Hadoop*, incluindo integração com o sistema de arquivos *HDFS*. O *Spark* é um framework que possibilita a execução das tarefas paralelizáveis de forma distribuída em máquinas multi-core ou clusters YARN/Mesos, com ênfase no processamento em *pipeline* de atividades, com alocação de arquivos intermediários primordialmente em memória. Códigos podem ser escritos em uma das linguagens para as quais o framework oferece sua API, incluindo: Scala, Java, Python e R. O framework possui arquitetura modular que permite a inclusão de componentes adicionais, tais como: Spark SQL, GraphX, Spark MLlib e Spark Streaming. Finalmente, o armazenamento de arquivos intermediários em memória faz com que a execução ocorra dezenas ou até centenas de vezes mais rápida que uma implementação equivalente em Apache *Hadoop*.

O *Spark* é constituído por cinco componentes principais, dos quais podem-se destacar: *Spark Core* e o *Spark SQL*. O *Spark Core* implementa a base do mecanismo de execução de tarefas, separando as *transformações* das *ações* sobre os *Resilient Distributed Datasets* (RDD) [Zaharia et al., 2012] e, mais recentemente, sobre os *dataframes*. As operações de *transformação* são de avaliação tardia e podem ser invocadas várias vezes antes que uma *ação* seja executada. Tal abordagem permite que o *Spark* escolha a melhor forma de executar as transformações considerando a quantidade de memória, os dados de localidade, o número de computadores no cluster, o número de cores nos processadores e as otimizações de código por meio do *Catalyst* [Armbrust et al., 2015]. Já o *Spark SQL* faz também uso do *Catalyst* para otimizar geração de código em tempo de execução à partir de expressões e comandos SQL. O *Spark SQL* permite acesso a datasets usando uma interface baseada em álgebra relacional [Armbrust et al., 2015]. Trata-se de um módulo para

processamento de dados estruturados. As interfaces do *Spark SQL* fornecem informações adicionais sobre a estrutura dos dados e a computação que está sendo executado. Com o *Spark SQL*, pode-se fazer uso, no mesmo programa, da linguagem SQL declarativa e APIs de acesso a dados em *DataFrame*/*Dataset* de modo imperativo/procedural.

4. Proposta do Workflow Framework W_fF

Esta seção descreve a proposta do framework W_fF. O diagrama UML conceitual da Figura 1.a apresenta a arquitetura do W_fF na perspectiva da execução de um workflow e suas ligações com a proveniência retrospectiva [Freire et al., 2008]. A arquitetura pode ser sumarizada por meio de suas nove classes principais que implementam os principais conceitos da álgebra de workflows [Ogasawara et al., 2011].

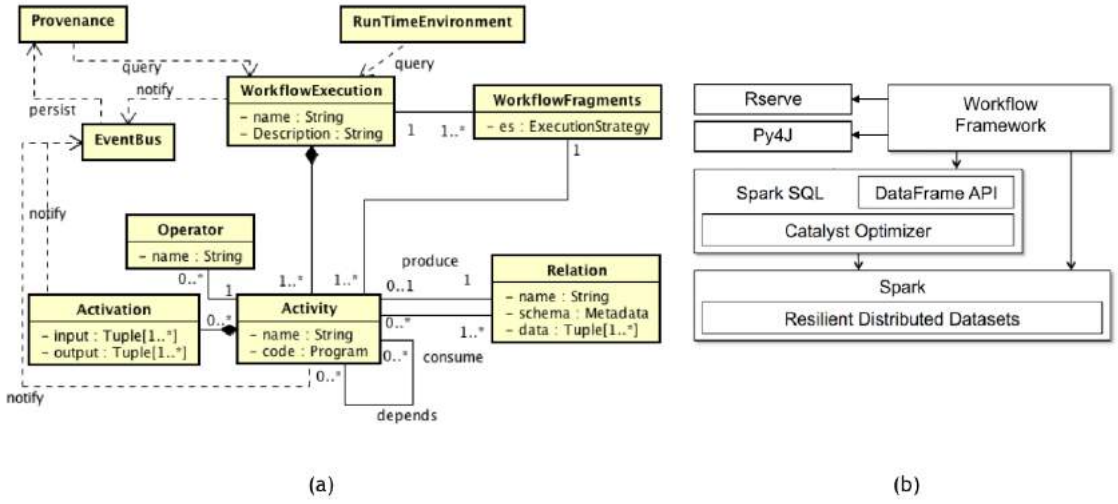

Figura 1. Diagrama UML conceitual descrevendo os componentes

A classe *WorkflowExecution* representa a execução de um workflow. O workflow é composto de atividades (*Activity*) que apresentam um programa (*Program*) associado. Cada atividade tem associada relações (*Relation*) de entradas e saída. A relação apresenta nome, esquema e os dados, propriamente ditos, representados por um conjunto de tuplas (*Tuple*). Além disto, as atividades têm um operador (*Operator*) da álgebra de workflows associado. O operador rege como a atividade deve consumir e produzir tuplas durante a sua execução por meio das ativações (*Activation*). Cada ativação consome um conjunto de tuplas e produz um conjunto de tuplas. Quando todas as ativações de uma atividade são processadas, tem-se a formação completa da relação de saída da atividade. A relação de produção e consumo das atividades produz uma dependência entre elas.

O W_fF está sendo projetado e escrito para funcionamento na linguagem Scala. Tal escolha visa a uma integração nativa com o *Spark* e seu componente de otimização (*Catalyst*). A classe *RuntimeEnvironment* abstrai o tipo de tecnologia a ser usada na execução do workflow. Ademais, em termos de apoio a execução de Programas, como o foco são workflows de análise de dados, além da execução de aplicações console, há também especializações específicas para execuções de atividades escritas em R e Python. A partir das características de projeto, pode-se apresentar com maior clareza os aspectos relacionados a otimização para execução do workflow.

Na arquitetura proposta, as relações, por exemplo, são especializações de *Dataset* do *Spark*. Elas podem tanto permanecer em memória *RAM* usando o cache do *Spark* ou serem materializadas. Como a linguagem Scala viabiliza métodos que sejam transformações (ver seção 3), as relações, são processadas em avaliação tardia. Tal característica viabiliza que o código seja otimizado de modo análogo ao processo de otimização de consultas em banco de dados.

Usando as informações de proveniência [Simmhan et al., 2005] de execuções anteriores, as informações do ambiente de execução (*RunTimeEnvironment*) e as próprias características intrínsecas das atividades, tem-se um processo de otimização que produz fragmentos de workflows (*WorkflowFragment*). Assim, no modelo proposto, cada fragmento de workflow é composto de um conjunto atividades. Os fragmentos têm um modelo de execução estabelecido (FAF ou FTF [Ogasawara et al., 2011]). Cada fragmento de workflow encapsula chamadas em Spark (Figura 1.b). As ativações, por consequência, são instanciadas e agrupadas segundo a estratégia execução associada aos seus respectivos fragmentos de workflow. Apenas as relações finais de cada fragmento de workflow são materializadas e têm as suas proveniências registradas. As demais, existem apenas no escopo dos *pipelines* formados no âmbito do modelo de execução FTF. Um exemplo didático deste processo é apresentado na seção 5.

O Diagrama UML da Figura 1.a também ilustra o barramento de eventos (*EventBus*). Neste barramento, os eventos são segregados e despachados por domínio para flexibilizar o controle e eficiência de execução. Dá-se a opção de instanciação de objetos polimórficos que podem compartilhar o acesso a eventos. A classe mais apropriada pode tratar o evento. Em termos de armazenamento da proveniência, por exemplo, cada ativação notificam o barramento de eventos, que delega a gravação da proveniência [Simmhan et al., 2005; Freire et al., 2008] a partir do seu *handler*.

No que tange às atividades, cada programa associado tem a opção de produzir eventos *onFinish* e *onError* engatilhado (do inglês, *triggered*) pelo W_fF. O tratamento de erro e steering para o usuário podem ser realizados no barramento por *handlers*. Finalmente, o barramento também apoia informações de notificações associadas à otimização, viabilizando *handlers* específicos para essa finalidade. Usando a API *GraphX* do *Spark*, gera-se, por exemplo, uma representação gráfica do workflow, com sua decomposição em fragmentos, em imagens, para fins de documentação ou depuração.

5. Discussão

Para fins de discussão, esta seção apresenta um workflow de *ETL* voltado a análise de séries espaço-temporais aplicada em problema de mobilidade urbana. O ETL faz a importação, limpeza e transformações nos dados obtidos de GPS dos ônibus do Rio de Janeiro coletados a cada minuto. Estes dados são fornecidos pela prefeitura da cidade. O período de análise engloba a Copa do Mundo de 2014 que aconteceu entre 12 de junho a 13 de julho. O intervalo de análise começa em 01/06/2017 e termina em 31/07/2014.

A álgebra de workflows [Ogasawara et al., 2011] descreve seis operadores algébricos. No contexto do W_fF, optou-se por simplificar e fazer uso apenas de cinco deles: *Map*, *SplitMap*, *Reduce*, *Filter* e *JoinQuery*. Este último é chamado apenas por *Query* visando uma integração mais estreita com SparkSQL [Armbrust et al., 2015].

A Figura 2.a ilustra a especificação do workflow usando a linguagem Scala. Neste

ETL de exemplo, o workflow faz uso apenas de três operadores: (i) $SplitMap$, (ii) Map, (iii) $Query$. No $SplitMap$, representado por $T \leftarrow SplitMap(Y, R, a)$, tem-se que a atividade Y produz um conjunto de tuplas na relação de saída T para cada tupla consumida na relação de entrada R. O conjunto de atributos a identifica as composições das tuplas produzidas na operação de *split*. Na linha 6 do exemplo, para o intervalo estabelecido na relação *trajectory*, são criadas 61 tuplas para coleta dos dados de mobilidade dos ônibus correspondente a cada dia do período de 1 de junho de 2014 a 31 de julho de 2014.

As operações envolvendo Map, seguem a notação $T \leftarrow Map(Y, R)$, onde a atividade Y produz uma única tupla na relação de saída T para cada tupla consumida na relação de entrada R. Isto ocorre nas linhas de 7 a 10 e na linha 12 do workflow. Na linha 7, para cada dia, faz-se o download efetivo dos dados de trajetória dos ônibus. Trata-se de um arquivo zip contendo um arquivo JSON por minuto. Em cada JSON, tem-se a localização de todos os ônibus em transito no Rio de Janeiro. Na linha 8, cada arquivo zip é processado e produz-se um arquivo RData com as observações minuto a minuto de todos os ônibus para aquele dia. Esta atividade remove os registros duplicados. Na linha 9, faz-se uma remoção de *outliers* por distribuição (velocidade instantânea, posicionamento e distância percorrida). A linha 10 produz as estações virtuais, considerando-se um raio para agrupamentos dos pontos de ônibus, a partir da malha de pontos de ônibus da cidade do Rio de Janeiro [Silva et al., 2016].

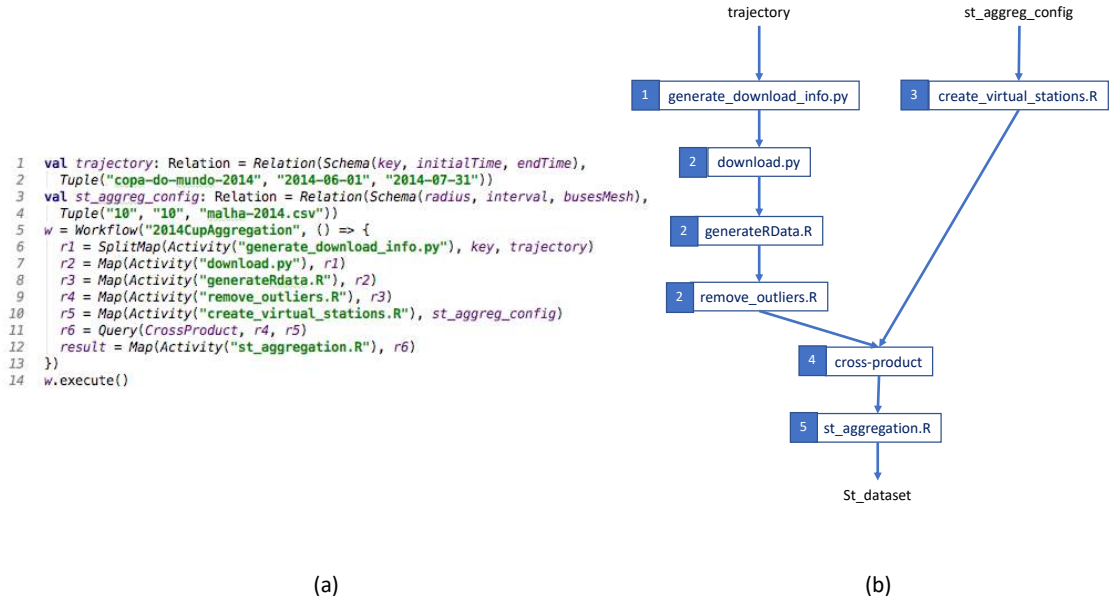

Figura 2. Workflow para análise de tráfego durante a COPA de 2014 : a) Especificação do Workflow usando linguagem Scala; b) grafo mostrando as dependências entre as atividades

A operação de $Query$, representada por $T \leftarrow Query(Y, R_1, ..., R_n)$, consome um conjunto de relações para produzir uma única relação de saída T. A atividade Y descreve uma consulta sobre as relações de entrada $\{R_1, ..., R_n\}$. Na linha 11, por exemplo, faz-se o produto cartesiano da relação r_5 com a relação r_4, *i.e.*, faz-se uma combinação entre os arquivos $RData$ preprocessados e as opções de configurações para agregações espaço-temporais [Silva et al., 2016].

Posteriormente, na linha, 12, tem-se a execução da agregação espaço-temporal em cada configuração de estações virtuais previamente estabelecida. Desta forma, produz-se séries espaço-temporais agregadas temporal e espacialmente, de acordo com as opções previstas na relação st_aggreg_config. Ao final do exemplo apresentado, são produzidos 61 arquivos RData de agregações espaço-temporais, um para cada dia.

A Figura 2.b apresenta a relação de dependência no workflow por meio de fluxo de dados entre as atividades. Pode-se observar que o workflow é constituído de duas relações de entrada independentes ($trajectory$ e st_aggreg_config). No ramo da esquerda, tem-se $trajetory$ com schema definido por *key, initialTime, endTime*. No exemplo, por simplicidade, esta relação possui apenas uma tupla que especifica os valores dos parâmetros para o operador *SplitMap* que produz a relação de saída r_1. Esta relação serve de entrada para a atividade "generate_download_info.py" e assim por diante.

No ramo da direita, tem-se st_aggreg_config com o schema definido por *radius, interval, busesMesh*. Da mesma forma, esta relação, por simplicidade, também apresenta apenas uma tupla onde especifica os parâmetros para criação das estações virtuais tendo r_5 como relação de saída. Este operador está associado a atividade "create_virtual_stations.R". Depois, faz-se uma operação de *Query* que realiza o produto cartesiano ligando os dois ramos e processa-se as agregações espaço-temporais para cada dia e configuração de agregação. O produto final é a relação $st\_dataset$.

Finalmente, as atividades indicadas na Figura 2.b, têm, ao lado esquerdo do seu nome, a indicação do número do fragmento de workflow associado. Ao todo, são cinco fragmentos. Os fragmentos que contêm apenas uma única atividade apresentam estratégia de execução do tipo FAF, enquanto que as atividades do fragmento 2 apresentam a estratégia de execução FTF [Ogasawara et al., 2011]. O processo de otimização inclui transformações algébricas que poderiam levar a mudanças na ordem de execução das atividades no workflow, mas que neste exemplo, não foi explorado.

6. Considerações finais

O presente trabalho apresenta uma proposta de implementação da álgebra de workflows em Apache Spark no apoio a processos de análise de dados. O trabalho procura ligar as vantagens de décadas de pesquisa na área de workflows com as ferramentas de análise de dados desenvolvidas recentemente e bastante difundidas nos últimos anos. Há também um amplo compromisso em apoio a existência de inúmeros projetos desenvolvidos em diversas linguagens usadas no contexto da Ciência de Dados, como Python e R e, ao mesmo tempo, apoiar aplicações legadas baseadas em programas de linha de comando.

Na implementação da prova de conceito, utilizou-se *Design Patterns*, tais como *Template Method*, *Command*, *Strategy*, *Proxy* e *ValueObject*, adaptados para a linguagem Scala. Como trabalhos futuros, pretendem-se realizar avaliações experimentais de speed-up e eficiência da solução em ambientes distribuídos de larga escala e medir a sobrecarga associada a gestão e armazenamento da proveniência e compará-las com soluções típicas de workflows científicos como, por exemplo, Chiron/Scicumulus [Ogasawara et al., 2013].

Referências

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A., et al. (2015). Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM.

CRAN (2017). The comprehensive r archive network. Technical report, https://cran.r-project.org.

Deelman, E., Gannon, D., Shields, M., and Taylor, I. (2009). Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540.

Fayad, M. and Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38.

Freire, J., Koop, D., Santos, E., and Silva, C. T. (2008). Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21.

Jergler, M., Sadoghi, M., and Jacobsen, H.-A. (2015). D2worm: A Management Infrastructure for Distributed Data-centric Workflows. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1427–1432, New York, NY, USA. ACM.

Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing*, 13(4):457–493.

Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., and Mattoso, M. (2011). An algebraic approach for data-centric scientific workflows. *Proc. of VLDB Endowment*, 4(12):1328–1339.

Ogasawara, E., Dias, J., Silva, V., Chirigati, F., de Oliveira, D., Porto, F., Valduriez, P., and Mattoso, M. (2013). Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16):2327–2341. 00042.

Rheinländer, A., Heise, A., Hueske, F., Leser, U., and Naumann, F. (2015). SOFA: An extensible logical optimizer for UDF-heavy data flows. *Information Systems*, 52:96–125.

Salloum, S., Dautov, R., Chen, X., Peng, P. X., and Huang, J. Z. (2016). Big data analytics on apache spark. *International Journal of Data Science and Analytics*, pages 1–20.

SciPy (2017). Python-based ecosystem of open-source software for mathematics, science, and engineering. Technical report, https://www.scipy.org.

Silva, A. B., Porto, F., and Ogasawara, E. (2016). Identificação de Motifs em Agregações de Séries Espaço-Temporais de Mobilidade Urbana. In *WTDBD*, Salvador, BA.

Simmhan, Y. L., Plale, B., and Gannon, D. (2005). A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36.

Van Der Aalst, W. M., Ter Hofstede, A. H., and Weske, M. (2003). Business process management: A survey. In *International conference on business process management*, pages 1–12. Springer.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2.