Análise *Online* de Dados de Proveniência e de Domínio de Aplicações Spark com SAMbA*

Thaylon Guedes[1], Vítor Silva[2], Marcos V. N. Bedo[1], Marta Mattoso[2], e Daniel de Oliveira[1]

[1]Universidade Federal Fluminense (UFF)

{thaylongs, marcosbedo}@id.uff.br, danielcmo@ic.uff.br

[2]Universidade Federal do Rio de Janeiro (COPPE/UFRJ)

{silva,marta}@cos.ufrj.br

Resumo. *Usuários dos mais diversos domínios de aplicação são capazes de se beneficiar de frameworks para processamento paralelo, escalável e intensivo em dados como o Apache Spark. Entretanto, o Apache Spark não oferece apoio à captura e gerência de dados de proveniência que auxiliam na depuração, reprodutibilidade e análise dos resultados. Um dos desafios em adicionar dados de proveniência ao Spark é capturar dados de proveniência distribuídos em estruturas em memória (RDDs) e associá-los ao histórico de derivação de dados. Esta demonstração apresenta o SAMbA, uma extensão para o Spark capaz de coletar dados de proveniência dos RDDs em tempo de execução e fornecer recursos analíticos ao usuário. Na demonstração utilizamos como estudo de caso uma execução de consultas por similaridade de forma paralela.*

1. Introdução

Muitas aplicações (científicas e comerciais) são compostas por operações que produzem dados de forma intensiva, como cargas, transformações e agregações de dados [Liu et al. 2016]. O fluxo dos dados gerados pelo encadeamento dessas operações é geralmente representado por meio de grafos direcionados acíclicos ou DAGs (do inglês *Directed Acyclic Graph*), no qual os vértices representam transformações de dados enquanto os arcos representam as dependências de dados entre elas. Várias aplicações consomem e produzem um grande volume de dados e adotam *scripts* para implementar tais DAGs. Entretanto, uma vez que essas aplicações necessitam processar conjuntos de dados de grande cardinalidade, o processamento paralelo, distribuído e escalável se torna essencial para o desempenho da aplicação [Jha et al. 2014].

Uma maneira simplificada de transformar *scripts* sequenciais em paralelos é por meio do uso de *frameworks* DISC (do inglês *Data Intensive Scalable Computing*). O Apache Spark[1] é um DISC que provê execução paralela e escalável de *scripts* com uso intensivo de dados. O Spark é baseado no processamento distribuído de dados em memória, por meio de uma abstração chamada RDD (do inglês *Resilient Distributed Dataset*). Os RDDs são coleções distribuídas de elementos de dados imutáveis. Uma vez modelada a aplicação, o Spark analisa o fluxo de dados da mesma para coordenar a execução paralela de instruções nas partições de dados que compõem um RDD.

*Os autores gostariam de agradecer a CAPES, CNPq e FAPERJ por financiarem parcialmente o trabalho

[1]https://spark.apache.org

Um exemplo de aplicação que requer processar grandes volume de dados é a execução de consultas por similaridade em espaços métricos. Esse tipo de consulta envolve, por exemplo, uma classificação baseada em distância ou recuperação de dados por conteúdo. Na prática, os dois tipos de consultas por similaridade mais utilizadas são as buscas por abrangência e vizinhança [Hetland 2009]. A modelagem e representação de operadores de buscas por similaridade em espaços métricos vem sendo estudada por muitos anos [Padmanabhan and Deshpande 2015] e diferentes estratégias de indexação têm sido propostas para otimizar o desempenho de sua execução, tais como os índices VP-Trees e VP-Forests. Estes índices são evoluções de árvores binárias e visam a executar buscas por similaridade com, potencialmente, complexidade logarítmica de tempo [Yianilos 1998]. Entretanto, a depender da cardinalidade e dimensionalidade do conjunto de dados de entrada, a quantidade de nós-folhas a ser examinada pode ser não-negligenciável, o que torna a execução da busca custosa. Portanto, consultas por similaridade (mesmo indexadas) podem se beneficiar do processamento paralelo do Spark.

Apesar de escalável para o processamento de grandes quantidades de dados, o Spark carece de apoio à proveniência [Freire et al. 2008]. Dados de proveniência são fundamentais para registrar a linhagem da geração de dados e facilitar a reprodutibilidade e a análise de dados produzidos por uma aplicação. Quando os dados de proveniência são enriquecidos com os dados do domínio da aplicação, eles se tornam uma base de dados importante para a análise *online* e *post-mortem*. Em execuções de longa duração, as consultas *online* são essenciais para monitorar e analisar o andamento da aplicação. O Spark fornece uma capacidade de proveniência limitada que registra as atividades já executadas em forma de arquivos log, mas é incipiente no que tange a prover apoio à reprodutibilidade e à análise de dados. Além disso, a depuração do código do Spark também é bastante difícil, dada a sua característica paralela e distribuída. Sistemas como `BigDebug` [Gulzar et al. 2016] e `Titian` [Interlandi et al. 2015] surgiram para fornecer recursos de depuração ao Spark, mas se restringem ao registro de dados de execução sem se preocuparem com a análise dos dados.

Este artigo tem por objetivo apresentar o SAMbA[2] (Spark provenAnce MAnagement), uma extensão para tornar o Spark capaz de coletar dados de proveniência a partir dos RDDs e fornecer recursos analíticos em tempo de execução. Samba supera os diversos desafios com relação à captura e registro de dados de proveniência no Spark, sendo um deles a de capturar dados de proveniência distribuídos nos RDDs e associá-los em memória ao caminho de derivação de dados de proveniência. Outro desafio relevante ocorre quando as transformações de dados no Spark trocam dados por meio de leitura e gravação de dados em arquivos. Nesse contexto, a transferência de elementos de dados entre atividades é implícita e o Spark não pode usar os RDDs para gerenciar esses dados porque não está ciente do fluxo de dados. Portanto, capturar os dados desses arquivos tem imenso potencial de melhorar a análise de dados de proveniência, mas também é um desafio [Freire et al. 2008, Silva et al. 2017].

2. Arquitetura do SAMbA

A captura e a gerência dos dados de proveniência de aplicações Spark não é uma tarefa simples. Primeiramente, os usuários devem ser capazes de definir o conteúdo apropriado

[2]`https://github.com/UFFeScience/SAMbA`

(elementos de dados de interesse) a serem extraídos dos RDDs. Além disso, aplicações Spark são geralmente executadas em ambientes DISC (nuvens ou *clusters*) e os usuários não têm ciência de qual máquina executará cada atividade. O SAMbA aborda ambos os desafios, coletando proveniência prospectiva e retrospectiva [Freire et al. 2008], além de informações do ambiente de execução e dos dados específicos do domínio. A proveniência prospectiva representa as operações que são executadas sobre os dados na aplicação, ao passo que a proveniência retrospectiva representa o *log* de execução da aplicação. O registro das informações do ambiente descrevem características relacionadas ao ambiente DISC. Por fim, os dados específicos de domínio representam o conteúdo produzido pelas transformações (resultados intermediários) que precisam ser analisados em conjunto com os dados de proveniência.

A Figura 1 apresenta os principais componentes da arquitetura do SAMbA. Os componentes do Spark coloridos são estendidos por meio do SAMbA e funcionam conforme discutido na sequência: o componente *Spark Context* conecta-se ao Gerenciador de *clusters* (Figura 2). Assim, o Spark instancia *Executors*, que são processos que executam operações e armazenam dados nas máquinas do ambiente DISC. Depois o Spark envia o código da aplicação (arquivo executável) para os *Executors*. Em seguida, o *SparkContext* envia tarefas para os *Executors* para serem processadas. Desse modo, o SAMbA é executado em cada *Executor* do Spark para capturar os dados de proveniência e de domínio.

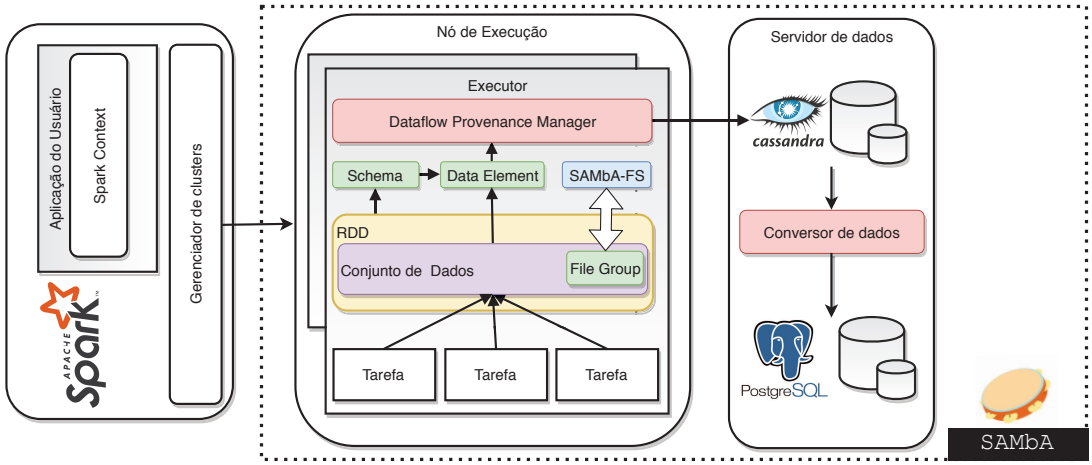

Figura 1. Arquitetura do SAMbA.

```
/* Spark Context */
val sparkConf = new SparkConf().setAppName("RangeQueryVpTree").setMaster("local[4]")
val sc = new SparkContext(sparkConf)
val forest: Broadcast[util.List[Node[Array[Double]]]] = sc.broadcast(rangeArgs.getArgsParser.getTrees)
```

Figura 2. Definição do *Spark Context* no SAMbA

O fluxo de um nó em execução na Figura 1, gerenciado pelo *Dataflow Provenance Manager*, registra a proveniência prospectiva e captura a proveniência retrospectiva do *dataflow* gerada pela execução da aplicação Spark. Essa captura de dados é baseada na extração de dados do RDD, através do *Data Element*, o qual representa cada unidade de dados que compõem o conteúdo de um RDD. Para adicionar dados de domínio à proveniência do *dataflow*, o SAMbA permite que o usuário defina atributos de interesse para

serem extraídos usando um componente chamado *Schema*. Assim, o *Dataflow Provenance Manager* interage com o *Data Element*, aplicando sobre ele o *Schema* (Figura 3).

```
.setSchema(new DataElementSchema[(RangeQueryParameter, String, Double)] {
  override def getFieldsNames(): Array[String] = Array("Target Element Id", "Candidate Element ID", "Distance")
```

Figura 3. Exemplo de uso do *Schema* no SAMbA.

O SAMbA concentra-se no usuário que desenvolveu a aplicação Spark, e que também conhece os dados do domínio. Assim, o *Schema* é uma *interface* com os métodos `getFieldsNames()`, que define os atributos a serem representados no *Schema*, e `splitData()`, que retorna uma coleção de dados (*Data Collection*) extraída do RDD. Um *Data Collection* é um conjunto de *Data Elements*, onde cada *Data Element* possui um identificador único (ID). Este ID é usado pelo SAMbA para rastrear a proveniência dentro do banco de dados que a armazena. Da mesma forma, cada operação executada no SAMbA também possui um ID, que é usado para associar o *Data Element* com as operações que o geraram.

Para resolver o problema de capturar dados de proveniência de programas caixa-preta, o SAMbA fornece o SAMbA-FS, um sistema de arquivos baseado em libfuse[3] (*Filesystem in Userspace*) do Linux, que mapeia um certo diretório do usuário para os dados representados por um RDD em memória. Como resultado, o Spark, e consequentemente o SAMbA, ficam cientes dos arquivo manipulados por programas caixa-preta. O SAMbA-FS utiliza o tipo de dado *FileGroup* que representa um conjunto de arquivos em memória. Para criar um RDD de *FileGroup*, utilizamos uma classe utilitária chamada *FileGroupTemplate*, e através dela, definimos quais arquivos devem ser carregados em memória – como exemplificado na Figura 4. Neste caso, o conteúdo do arquivo chamado *inputFastaList.txt* é carregado na memória (RDD). Além disso, o *FileGroupTemplate* permite armazenar um mapa de chave-valor, que serve para armazenar informações extras sobre o *FileGroup*, como o nome do arquivo ou algum parâmetro de configuração.

```
val fileGroupTemplate = FileGroupTemplate.ofFile(
      new File("/home/user/dataflow/inputFastaList.txt"),
      false, Map("FILE_NAME" -> "inputFastaList.txt")
)
val rdd = sparkContex.fileGroup(fileGroupTemplate)
```

Figura 4. Criação de um RDD de *FileGroup*.

Quando um RDD de *FileGroup* é criado, dois novos operadores de transformação são fornecidos pelo SAMbA: *runCommand*, que executa comandos nativos do SO, e *runScientificApplication* que executa programas ou *scripts* caixa-preta contendo sequência de invocações de programas (Figura 5) que se encontram em um determinado diretório.

```
rdd.runScientificApplication("someScript.sh {{FILE_NAME}}")
```

Figura 5. Exemplo da execução de um *script* como um parâmetro *FILE_NAME* proveniente do mapa de chave-valor do FileGroup.

[3] `https://github.com/libfuse/libfuse`

Somando-se a essas características, o SAMbA carrega dados de proveniência e domínio em seu banco de dados de forma *online*, ou seja, durante a execução de aplicações Spark. Desta forma, a proveniência e os dados de domínio são fornecidos para o usuário em tempo de execução, permitindo assim que os usuários executem consultas enquanto a aplicação ainda está em execução. Os dados de proveniência capturados são armazenados em um banco de dados Apache Cassandra pelo *Dataflow Provenance Manager*.

3. Demonstração

Em nossa demonstração, exemplificamos como o SAMbA pode ajudar os usuários a executar aplicações Spark e realizar análises *online* sobre os dados específicos de domínio armazenados em memória. Para tanto, utilizamos como estudo de caso uma aplicação que paraleliza consultas por similaridade indexadas em espaços métricos. A ideia principal desse estudo de caso é acelerar a avaliação dos elementos nos nós-folha de índices VP-Tree e VP-Forest ao resolver uma consulta por similaridade.

Nesse contexto, os nós-folha incluem os elementos de dados a serem consultados e são armazenados como arquivos sequenciais, enquanto que as estruturas das árvores em si são mantidas como arquivos separados. Dada uma consulta por abrangência com um raio de tolerância e um elemento de busca, definimos uma rotina Spark para selecionar os nós-folha que interceptam o raio da consulta e examinar a distância das instâncias cobertas pelo nó com relação ao elemento de consulta. Nesse cenário, exemplos de consultas *online* que podem ser executadas pelos usuário para acompanhar a execução são: (1) Verificar quais foram os nós-folhas que foram inspecionados, ou (2) Verificar quantos nós foram descartados por não incluírem elementos candidatos ao conjunto-resposta.

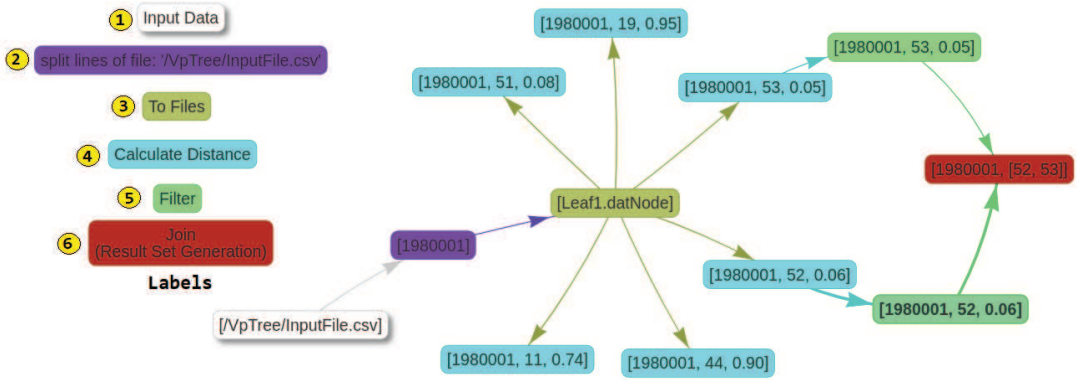

Figura 6. Grafo de proveniência do SAMbA para busca por abrangência em um nó-folha de um índice VP-Tree.

Além de executar essa aplicação no Spark, os usuários podem visualizar o grafo de proveniência gerado (Figura 6). Para este estudo de caso, usamos conjuntos de dados sintéticos de 10 e 100 dimensões comparados pela distância Euclideana e usando índices com diversas parametrizações no número de elementos por nó-folha. O grafo apresentado na Figura 6 representa a execução do dataflow para uma consulta por abrangência cujo o raio seja menor ou igual a 0,06 unidades. A aplicação do estudo de caso começa lendo o arquivo de entrada (1). Para cada elemento de consulta presente no arquivo de testes (2), são identificados os nós-folhas dos índices que talvez contenham elementos que atendam o critério de seleção. Cada elemento do nó-folha (3) é identificado pelo seu ID e tem a

sua distância calculada até o elemento de consulta. Em seguida, são selecionados aqueles cuja a distância é menor ou igual a 0,06 unidades (4). Por fim, a aplicação agrupa os identificadores dos elementos que passaram no teste (5). Para a apresentação do SAMbA no evento, incentivamos usuários a trazerem suas próprias aplicações Spark.

4. Conclusão

O SAMbA é uma extensão do Apache Spark para apoiar a análise *online* de dados de proveniência armazenados em memória. Nesta demonstração apresentamos um exemplo do uso do SAMbA para captura e consulta de proveniência *online* em uma aplicação de busca por similaridade. Como trabalhos futuros, pretende-se investigar o uso do SAMbA no gerenciamento de proveniência em aplicações científicas com relação a consultas *online* e *post-mortem*. A versão beta do SAMbA pode ser obtida no repositório `https://github.com/UFFeScience/SAMbA`. Além disso, está disponível no mesmo endereço do repositório, um vídeo de demonstração do sistema e um guia inicial para usuários que desejam utilizar o SAMbA.

Referências

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.

Gulzar, M. A., Interlandi, M., Yoo, S., Tetali, S. D., Condie, T., Millstein, T., and Kim, M. (2016). Bigdebug: Debugging primitives for interactive big data processing in spark. Em *Proceedings of the 38th International Conference on Software Engineering*, páginas 784–795. ACM.

Hetland, M. L. (2009). The basic principles of metric indexing. Em *Swarm Intelligence for Multi-objective Problems in Data Mining*, páginas 199–232. Springer.

Interlandi, M., Shah, K., Tetali, S. D., Gulzar, M. A., Yoo, S., Kim, M., Millstein, T., and Condie, T. (2015). Titian: Data provenance support in spark. *Proceedings of the VLDB Endowment*, 9(3):216–227.

Jha, S., Qiu, J., Luckow, A., Mantha, P., and Fox, G. C. (2014). A tale of two data-intensive paradigms: Applications, abstractions, and architectures. Em *IEEE International Congress on Big Data*, páginas 645–652.

Liu, J., Pacitti, E., Valduriez, P., de Oliveira, D., and Mattoso, M. (2016). Multi-objective scheduling of scientific workflows in multisite clouds. *Future Generation Computer Systems*, 63:76 – 95. Modeling and Management for Big Data Analytics and Visualization.

Padmanabhan, D. and Deshpande, P. M. (2015). *Operators for Similarity Search - Semantics, Techniques and Usage Scenarios*. Springer.

Silva, V., Leite, J., Camata, J. J., de Oliveira, D., Coutinho, A. L., Valduriez, P., and Mattoso, M. (2017). Raw data queries during data-intensive parallel workflow execution. *Future Generation Computer Systems*, 75:402 – 422.

Yianilos, P. N. (1998). Excluded middle vantage point forests for nearest neighbor search. Technical report, NEC Research Institute, Princeton, NJ.