Big Data com Apache Spark - Parte 1: Introdução

Lire ce contenu en français

O que é o Spark

O Spark é um framework para processamento de Big Data construído com foco em velocidade, facilidade de uso e análises sofisticadas. Está sendo desenvolvido desde de 2009 pelo AMPLab da Universidade de Califórnia em Berkeley e em 2010 seu código foi aberto como projeto da fundação Apache.

O Spark tem muitas vantagens se comparado as outras tecnologias de Big Data e do paradigma MapReduce, como o Hadoop e o Storm.

Inicialmente, o Spark oferece um framework unificado e de fácil compreensão para gerenciar e processar Big Data com uma variedade de conjuntos de dados de diversas naturezas (por exemplo: texto, grafos, etc), bem como de diferentes origens (batch ou streaming de dados em tempo real).

O Spark permite que aplicações em clusters Hadoop executem até 100 vezes mais rápido em memória e até 10 vezes mais rápido em disco,
desenvolver rapidamente aplicações em Java, Scala ou Python. Além disso, vem com um conjunto integrado de mais de 80 operadores de alto nível e pode ser usado de forma interativa para consultar dados diretamente do console.

Além das operações de Map/Reduce, suporta consultas SQL, streaming de dados, aprendizado de máquina e processamento de grafos. Desenvolvedores podem usar esses recursos no modo stand-alone ou combiná-los em um único pipeline.

Nesta primeira edição dessa série de artigos sobre o Apache Spark, veremos o que é o Spark, como ele se compara com uma solução típica com MapReduce e disponibiliza um conjunto completo de ferramentas para processamento de Big Data.

Hadoop e Spark

O Hadoop já existe a mais de 10 anos e tem provado ser a melhor solução para o processamento de grandes conjuntos de dados. O MapReduce, é uma ótima solução para cálculos de único processamento, mas não muito eficiente para os casos de uso que requerem cálculos e algoritmos com várias execuções. Isso porque cada etapa no fluxo de processamento tem apenas uma fase Map e uma fase Reduce e, desse modo é necessário converter qualquer caso de uso para o padrão MapReduce para chegar a uma solução.

Os dados de saída do processamento de cada etapa devem ser armazenados no sistema de arquivo distribuídos antes do passo seguinte começar. Assim, esta abordagem tende a ser lenta devido à replicação e armazenamento em disco. Além disso, as soluções Hadoop incluem tipicamente clusters que são difíceis de configurar e gerenciar, além de precisar da integração de várias ferramentas para diferentes casos de uso de Big Data (como o Mahout para Aprendizado de Máquina e o Storm para o processamento de streaming).

Nesse cenário, caso seja necessário fazer algo complexo, seria preciso encadear uma série de jobs de MapReduce e executá-los em sequência. Cada um desses jobs terão alta latência e não poderá começar até que o anterior tenha terminado.

O Spark permite que os programadores desenvolvem pipelines compostos por várias etapas complexas usando grafos direcionais acíclicos. Além disso, suporta o compartilhamento de dados da memória através desses grafos, de modo que os diferentes jobs possam trabalhar com os mesmos dados.

O Spark usa a infraestrutura do Hadoop Distributed File System (HDFS), mas melhora suas funcionalidades e fornece ferramentas adicionais. Por exemplo, permite a implantação de aplicativos em cluster Hadoop v1 (com SIMR - Spark Inside MapReduce), ou em Hadoop v2 com YARN ou com Apache Mesos.

Devemos olhar para o Spark como uma alternativa para MapReduce do Hadoop em vez de um simples substituto, mas como uma solução abrangente e unificada para gerenciar diferentes casos de uso da Big Data.

Características do Spark

O Spark estende o MapReduce evitando mover os dados durante seu processamento, através de recursos como armazenamento de dados em memoria e processamento próximo ao tempo real, o desempenho pode ser várias vezes mais rápido do que outras tecnologias de Big Data.

Também há suporte para validação sob demanda de consultas para Big Data, o que ajuda com a otimização do fluxo de processamento de dados e fornece uma API de mais alto nível para melhorar a produtividade do desenvolvedor e um modelo consistente para o arquiteto de soluções Big Data.

O Spark detém resultados intermediários na memória, em vez de escrevê-los no disco, o que é muito útil quando se precisa processar o mesmo conjuntos de dados muitas vezes. Seu projeto teve por objetivo torná-lo um mecanismo de execução que funciona tanto na memória como em disco e, por isso, o Spark executa operações em disco quando os dados não cabem mais na memória. Assim, é possível usá-lo para o processamento de conjuntos de dados maiores que a memória agregada em um cluster.

O Spark armazenará a maior quantidade possível de dados na memória e, em seguida, irá persisti-los em disco. Cabe ao arquiteto do sistema olhar para os seus dados e casos de uso para avaliar os requisitos de memória. Com esse mecanismo de armazenamento de dados em memória, o uso do Spark traz vantagens de desempenho.

Outras características do Spark:

- Suporta mais do que apenas as funções de Map e Reduce;
- Otimiza o uso de operadores de grafos arbitrários;
- Avaliação sob demanda de consultas de Big Data contribui com a otimização do fluxo global do processamento de dados;
- Fornece APIs concisas e consistentes em Scala, Java e Python;
- Oferece shell interativo para Scala e Python. O shell ainda não está disponível em Java.

O Spark é escrito na linguagem Scala e executa em uma máquina virtual Java. Atualmente, suporta as seguintes linguagens para o desenvolvimento de aplicativos:

- Scala
- Java
- Python
- Clojure
- R

O ecossistema do Spark

Além da API do Spark, existem bibliotecas adicionais que fazem parte do seu ecossistema e fornecem capacidades adicionais para as áreas de análise de Big Data e aprendizado de máquina.

Estas bibliotecas incluem:

- **Spark Streaming:**
  - O Spark Streaming pode ser usado para processar dados de streaming em tempo real baseado na computação de microbatch. Para isso é utilizado o DStream que é basicamente uma série de RDD para processar os dados em tempo real;
- **Spark SQL:**
  - Spark SQL fornece a capacidade de expor os conjuntos de dados Spark através de uma API JDBC. Isso permite executar consultas no estilo SQL sobre esses dados usando ferramentas tradicionais de BI e de visualização. Além disso, também permite que os usuários usem ETL para extrair seus dados em diferentes formatos (como JSON, Parquet, ou um banco de dados), transformá-los e expô-los para consultas ad-hoc;
- **Spark MLlib:**
  - MLlib é a biblioteca de aprendizado de máquina do Spark, que consiste em algoritmos de aprendizagem, incluindo a classificação, regressão, clustering, filtragem colaborativa e redução de dimensionalidade;
- **Spark GraphX:**
  - GraphX é uma nova API do Spark para grafos e computação paralela. Em alto nível, o GraphX estende o Spark RDD para grafos. Para apoiar a computação de grafos, o GraphX expõe um conjunto de operadores fundamentais (por exemplo, subgrafos e vértices adjacentes), bem como uma variante optimizada do Pregel. Além disso, o GraphX inclui uma crescente coleção de algoritmos para simplificar tarefas de análise de grafos.

Além destas bibliotecas, outros componentes completam o ecossistema do Spark, como o BlinkDB e o Tachyon.

O BlinkDB é uma engine SQL para consultas por amostragem e pode ser usado para a execução de consultas interativas em grandes volumes de dados. Permite que os usuários equilibrem a precisão de consulta com o tempo de resposta. Além disso, o BlinkDB funciona em grandes conjuntos de dados, através de amostragem de dados e apresentação de resultados anotados com os valores de erros.

O Tachyon é um sistema de arquivos distribuídos em memória que permite o compartilhamento de arquivos de forma confiável e rápida através de frameworks de cluster, como Spark e MapReduce. Também armazena em cache os arquivos que estão sendo trabalhados, permitindo que a existência de diferentes processamentos / consultas e enquadramentos para acessar arquivos em cache na velocidade de memória.

Finalmente, há também adaptadores de integração com outros produtos, como Cassandra (Cassandra Spark Connector) e R (SparkR). Com o Cassandra Connector, é possível usar o Spark para acessar dados armazenados no banco de dados Cassandra e realizar com o R análises estatísticas.

O diagrama a seguir (Figura 1) mostra como as diferentes bibliotecas do ecossistema Spark estão relacionados uns com os outros.

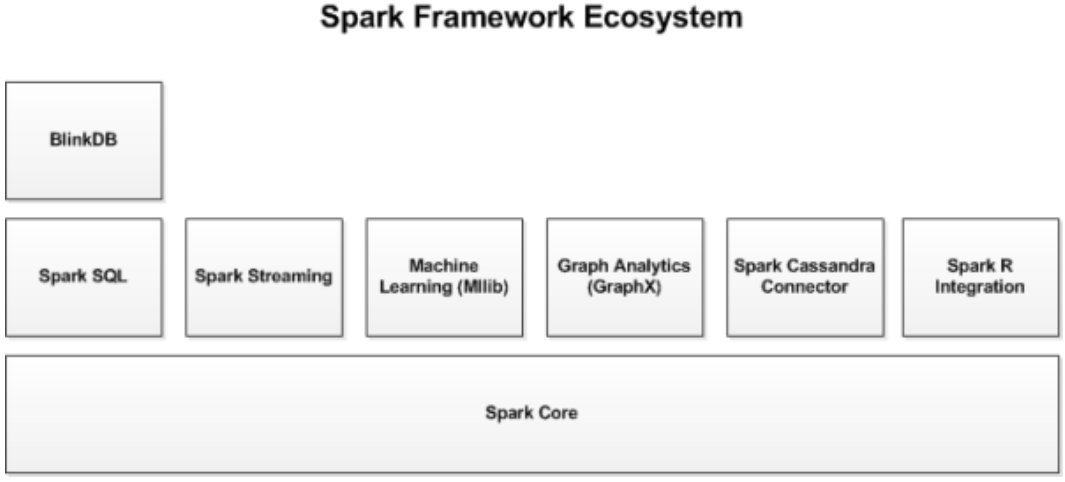

Figura 1. Bibliotecas do Framework Spark.

Vamos explorar todas essas bibliotecas nos próximos artigos desta série.

A arquitetura do Spark

A arquitetura Spark inclui os seguintes componentes:

- Armazenamento de dados;
- API;
- Framework de gerenciamento.

Vejamos cada um desses componentes em detalhes.

Armazenamento de dados:

O Spark usa sistema de arquivos HDFS para armazenamento de dados. Funciona com qualquer fonte de dados compatível com Hadoop, incluindo o próprio HDFS, HBase, Cassandra, etc.

API:

A API permite que os desenvolvedores de aplicações criem aplicações baseadas no Spark usando uma interface de API padrão para Scala, Java e Python.

A seguir, estão os links dos site da API Spark para as linguagens: Scala, Java e Python

Gestão de recursos:

O Spark pode ser implantado como um servidor autônomo ou em uma estrutura de computação distribuída como o Mesos ou o YARN. Na Figura 2, apresentam-se os componentes da arquitetura Spark.

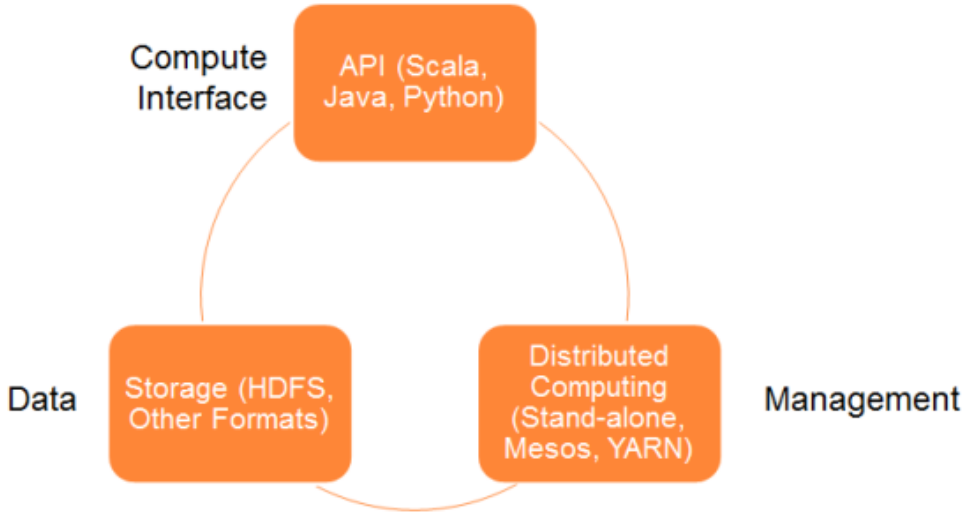

Figura 2. Arquitetura do Spark.

Conjunto de dados resilientes e distribuídos

O conjunto de dados resilientes e distribuídos (base do trabalho de pesquisa de Matei Zaharia) ou RDD (Resilient Distributed Datasets) é o conceito central do framework Spark. Imagine o RDD como uma tabela do banco de dados que pode guardar qualquer tipo de dado. O Spark armazena os dados do RDD em diferentes partições. Isso ajuda a reorganização computacional e a otimização no processamento dos dados.

Os RDDs são imutáveis. Ainda que aparentemente seja possível modificar um RDD com uma transformação, na verdade o resultado dessa transformação é um novo RDD, sendo que o original permanece intocável.

O RDD suporta dois tipos de operações:

Transformação: Não retornam um único valor, mas um novo RDD. Nada é avaliado quando a função de transformação é chamada, ela apenas recebe um RDD e retorna um novo RDD.

Algumas das funções de transformação são map, filter, flatMap, groupByKey, reduceByKey, aggregateByKey, pipe e coalesce.

Ação: Esta operação avalia e retorna um novo valor. Quando uma função de ação é chamado em um objeto RDD, todas as consultas de processamento de dados são computadas e o valor é retornado. Algumas das operações de ação são reduce, collect, count, first, take, countByKey e foreach.

Como instalar o Spark

Existem algumas maneiras de instalar e usar Spark: É possível instalá-lo em sua máquina para execução stand-alone ou usar uma máquina virtual (VM) disponibilizadas por fornecedores como Cloudera, Hortonworks ou MapR. Ou também é possível utilizar um Spark instalado e configurado na nuvem (como na Databricks Cloud).

Neste artigo, vamos instalar Spark como um framework stand-alone e executá-lo localmente. Vamos usar a versão 1.2.0 para o código da aplicação exemplo.

Como executar o Spark

Durante a instalação do Spark em máquina local ou na nuvem, existem diferentes maneiras nas quais é possível acessar o engine do Spark.

A tabela a seguir mostra como configurar o parâmetro Master URL para diferentes modos de funcionamento do Spark.

Master URL	Description
Local	Run Spark locally with one worker thread (i.e. no parallelism at all).
local[K]	Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
local[*]	Run Spark locally with as many worker threads as logical cores on your machine.
spark://HOST:PORT	Connect to the given Spark standalone cluster master. The port must be whichever one your master is configured to use, which is 7077 by default.
mesos://HOST:PORT	Connect to the given Mesos cluster. The port must be whichever one your is configured to use, which is 5050 by default. Or, for a Mesos cluster using ZooKeeper, use mesos://zk://....
yarn-client	Connect to a YARN cluster in client mode. The cluster location will be found based on the HADOOP_CONF_DIR variable.
yarn-cluster	Connect to a YARN cluster in cluster mode. The cluster location will be found based on HADOOP_CONF_DIR.

Como interagir com o Spark

Uma vez que o Spark esteja instalado e funcionando, é possível conectar nele usando um shell para análise interativa dos dados. O Spark Shell está disponível em Scala e Python.

Para acessá-los, execute respectivamente os comandos spark-shell.cmd e pyspark.cmd.

Console Web do Spark

Independente do modo de execução, é possível visualizar os resultados e outras estatísticas através do Web Console disponível na URL:

```
http://localhost:4040
```

O Console do Spark é mostrado na Figura 3 a seguir com abas para Stages, Storage, Ecosystem e Executor.

(Clique na imagem para ampliá-la)

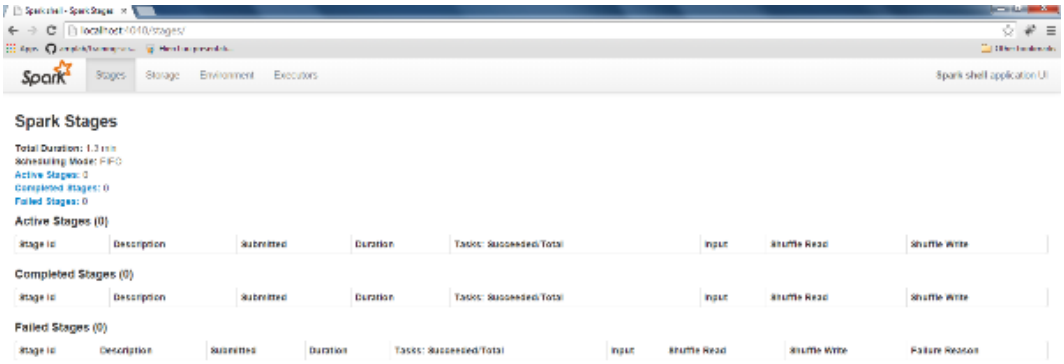

Figura 3. Console Web do Spark.

Variáveis compartilhadas

O Spark oferece dois tipos de variáveis compartilhadas para torná-lo eficiente para execução em cluster. Estas variáveis são dos tipos Broadcast e Acumuladores.

Broadcast: ou variáveis de difusão, permitem manter variáveis somente leitura no cache de cada máquina em vez de enviar uma cópia junto com as tarefas. Essas variáveis podem ser usadas para dar aos nós do cluster as cópias de grandes conjuntos de dados.

O seguinte trecho de código mostra como usar as variáveis de broadcast.

```
//
// Variáveis de Broadcast
//
val broadcastVar = sc.broadcast(Array(1, 2, 3))
broadcastVar.value
```

Acumuladores: permitem a criação de contadores ou armazenar os resultados de somas. As tarefas em execução no cluster podem adicionar valores à variável do acumulador usando o método *add*. No entanto, as tarefas distintas não podem ler o seu valor pois apenas o programa principal pode ler o valor de um acumulador.

O trecho de código a seguir mostra como usar criar um acumulador:

```
//
// Acumulador
//
val accum = sc.accumulator(0, "My Accumulator")
sc.parallelize(Array(1, 2, 3, 4)).foreach(x => accum += x)
accum.value
```

Amostra de uma Aplicação do Spark

A aplicação de exemplo deste artigo é um aplicativo de contagem simples de palavras. Este é o mesmo exemplo apresentado em muitos tutoriais sobre o Hadoop. Vamos realizar algumas consultas de análise de dados em um arquivo de texto. Esse arquivo e o conjunto de dados desse exemplo são pequenos, mas as mesmas consultas Spark podem ser usadas para grandes conjuntos de dados sem quaisquer modificações no código.

Para facilitar a apresentação, usaremos o shell de comandos para Scala.

Primeiramente, veremos como instalar Spark em sua máquina local.

Pré-requisitos:

- Será necessário instalar o Java Development Kit (JDK) para trabalhar localmente. A instalação da JDK é coberta no Etapa 1 a seguir.
- Também precisar instalar o software do Spark na sua máquina local. As instruções sobre como fazer isso são abordadas na Etapa 2 a seguir.

Nota: Estas instruções estão preparadas para ambiente o Windows. Se estiver usando um sistema operacional diferente, será necessário modificar as variáveis do sistema e caminhos de diretório de acordo com seu ambiente.

I. Instalar o JDK:

1) Faça download do JDK no site Oracle. A versão 1.7 do JDK é a recomendada.

Instale o JDK em um diretório sem espaços. Isto é, para usuários do Windows, instale o JDK em uma pasta similar a "C:\dev", e não em "C:\Arquivos de Programas". O diretório "Arquivos de Programas" tem um espaço no nome e isso pode causar problemas quando o software é instalado nesta pasta.

Nota: Não instale o JDK ou o Spark (descrito na Etapa 2) em "C:\Arquivos de Programas".

2) Depois de instalar o JDK, verifique se ele foi instalado corretamente navegando via prompt até a pasta "bin" dentro do diretório do JDK 1.7 e digitando o seguinte comando:

```
java -version
```

Se o JDK está instalado corretamente, o comando exibirá a versão instalada do Java.

II. Instalar o software do Spark:

Baixe a versão mais recente do Spark. A versão mais recente no momento da escrita deste artigo é Spark 1.2. É possível escolher uma instalação Spark específico, dependendo da versão do Hadoop. Baixei Spark para Hadoop 2.4 ou posterior, cujo nome do arquivo é spark-1.2.0-bin hadoop2.4.tgz.

Descompacte o arquivo de instalação em um diretório local (por exemplo: "C:\dev").

Para verificar a instalação o Spark, navegue até o diretório e execute o shell do Spark utilizando os comandos a seguir. Isto é para Windows. Se estiver usando Linux ou Mac OS, edite os comandos para trabalhar em seu sistema operacional.

```
c:
cd c:\dev\spark-1.2.0-bin-hadoop2.4
bin\spark-shell
```

Se o Spark foi instalado corretamente, será apresentado as seguintes mensagens na saída no console.

```
….
15/01/17 23:17:46 INFO HttpServer: Starting HTTP Server
15/01/17 23:17:46 INFO Utils: Successfully started service 'HTTP class server' on port 5
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/

Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71)
Type in expressions to have them evaluated.
Type :help for more information.
….
15/01/17 23:17:53 INFO BlockManagerMaster: Registered BlockManager
15/01/17 23:17:53 INFO SparkILoop: Created spark context..
Spark context available as sc.
```

Digitar os seguintes comandos para verificar se o Spark está funcionando corretamente:

```
sc.version
```

(ou)

```
sc.appName
```

Após esta etapa, pode sair da janela de shell do Spark digitando o seguinte comando:

```
:quit
```

Para iniciar o shell do Spark Python é preciso ter o Python instalado em sua máquina. Sugerimos o download e instalação do Anaconda, que é uma distribuição Python livre e inclui vários pacotes Python populares para matemática, engenharia e análise de dados.

Em seguida, execute os seguintes comandos:

```
c:
cd c:\dev\spark-1.2.0-bin-hadoop2.4
bin\pyspark
```

Aplicação Word Count

Uma vez que já tem o Spark instalado e funcionando, podemos executar as consultas de análise de dados usando API do Spark.

A seguir temos alguns comandos simples para ler os dados de um arquivo e processá-los. Vamos trabalhar com casos de uso avançados nos próximos artigos desta série.

Primeiro, vamos usar API Spark para realizar a contagem das palavras mais populares no texto. Abra uma nova shell do Spark e execute os seguintes comandos:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

val txtFile = "README.md"
val txtData = sc.textFile(txtFile)
txtData.cache()
```

Chamamos a função de cache para armazenar o RDD criado no passo anterior, então o Spark não tem de computá-lo em cada uso nas consultas posteriores. Note que cache() é uma operação lazy, portanto o Spark não armazena imediatamente os dados na memória. Na verdade, só será alocado se uma ação for chamada no RDD.

```
txtData.count()
```

Agora, podemos chamar a função contagem para ver quantas linhas existem no arquivo de texto.

```
val wcData = txtData.flatMap(l => l.split(" ")).map(word => (word, 1)).reduceByKey(_ + _)

wcData.collect().foreach(println)
```

Se quiser ver mais exemplos de códigos de uso Spark API Core, verifique a documentação Sparkem seu site.

Próximos passos

Nos próximos artigos desta série, vamos aprender mais sobre outras partes do Spark começando com Spark SQL. Após isso, veremos o Spark Streaming, o Spark MLlib, e o Spark GraphX. Também veremos frameworks como Tachyon e BlinkDB.

Conclusões

Neste artigo, abordamos como o framework Apache Spark ajuda no processamento e analise de Big Data com sua API padrão. Também comparamos o Spark com a implementação tradicional do MapReduce. O Spark é baseado no mesmo sistema de armazenamento de arquivos HDFS como o Haddop, assim é possível usar Spark e o MapReduce em conjunto. Isto é especialmente interessante se você já tem um investimento significativo e configuração de infraestrutura com o Hadoop.

Também é possível combinar o processamento Spark com o Spark SQL, Aprendizado de Máquina e o Apache Streamming.

Com várias integrações e adaptadores, é possível combinar o Spark com outras tecnologias. Um exemplo disto é a utilização de Spark, Kafka, e Apache Cassandra juntos, no qual o Kafka pode ser utilizada para entrada dos dados de streaming, o Spark para fazer processamento e, finalmente, armazenar no banco de dados Cassandra os resultados desse processamento.

Mas, tenha em mente que o Spark possui o ecossistema menos maduro e precisa de algumas melhorias áreas como segurança e integração com ferramentas de BI.

Referências

- Site principal do Spark
- Exemplos do Spark
- Apresentações e Vídeos da conferência Spark Summit 2014
- Site do Databricks

Sobre o autor

Srini Penchikala atualmente trabalha como Arquiteto de Software em uma organização de serviços financeiros em Austin, Texas. Ele tem mais de 20 anos de experiência em arquitetura, design e desenvolvimento. Srini está atualmente escrevendo um livro sobre padrões de bancos de dados NoSQL e também é co-autor do livro "Spring Roo in Action" da Manning Publications. Ele palestrou em diversas conferências como: JavaOne, SEI Architecture Technology Conference (SATURN), IT Architect Conference (ITARC), No Fluff Just Stuff, NoSQL Now e Project World Conference. Srini também publicou diversos artigos sobre arquitetura de software, segurança e gerenciamento de risco, e sobre banco de dados NoSQL em sites como o InfoQ, The ServerSide, OReilly Network (ONJava), DevX Java, java.net e JavaWorld. Ele é o Editor Líder de banco de dados no NoSQL na comunidade do InfoQ.