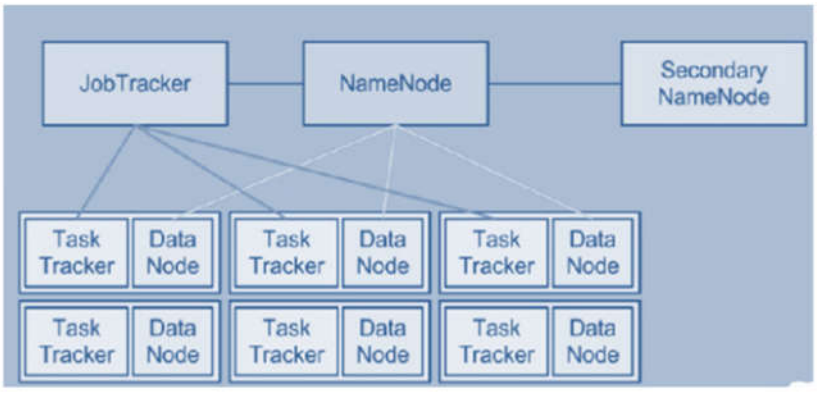
Soluções para processamento paralelo e distribuído de dados

Hadoop – Arquitetura do Hadoop 2.0: YARN

Ilustração de Oliver Munday

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

- Características

Name Node

- Todos os metadados são mantidos em RAM;
- Executar todas as operações relativas aos metadados;
- Ponto de falha da arquitetura (há condições de contorno);

Limitações da arquitetura

- Escalabilidade 'limitada';
- Utilização de recurso pode ser melhorada;
- Não há suporte para outras paradigmas de processamento;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

- Características

- Escalabilidade 'limitada'
  - ~ 4000 nós de dados
  - ~ 40000 tarefas concorrentes

- Utilização de recursos pode ser melhorada

Cluster Summary (Heap Size is 12.58 GB/23.97 GB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
4512	285	2223	188	4512	285	0	0	4512	3008	40.00

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

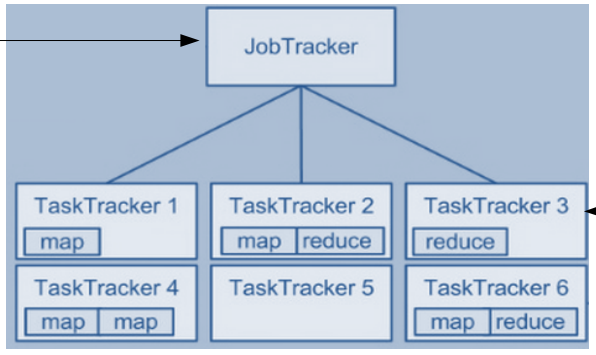
- Características
  - Utilização de recursos pode ser melhorada

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

- Características
  - Utilização de recursos pode ser melhorada

- Auditoria dos TaskTrackers
- Agendamento de jobs
- Monitoração das tarefas map e reduce



Executar tarefas map e reduce e retornar para o JobTracker

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

Características

- Tarefas do JobTracker

  Gerencia dos recursos computacionais(tarefas map e reduce);

  Agendamento de todas as tarefas de usuários:

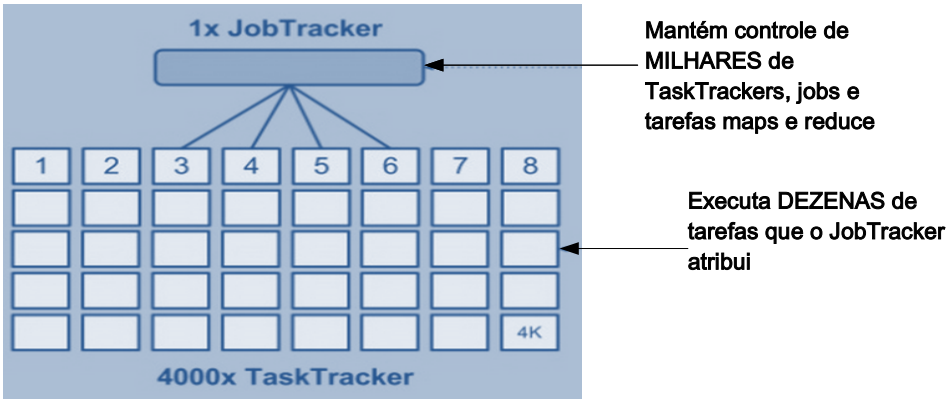
  - Agendar tarefas de um job;
  - Monitorar execução das tarefas;
  - Reiniciar tarefas que falharam e verificar recursos disponíveis;
  - Calcular e manter atualizados os registros/ contadores de jobs;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

Características

- Algumas observações



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 1.0

Características

- Propostas 'naturais'

  Reduzir as responsabilidades do JobTracker;

  - Separar o gerenciamento de recursos do cluster da coordenação de Jobs;
  - Usar outros nós para gerenciar o ciclo de vida dos jobs;

  Aumentar a escalabilidade

  - Mais de 10.000 nós;
  - Mais de 10.000 jobs;
  - Mais de 100.000 tarefas;

Hadoop - Arquitetura do Hadoop 2.0: YARN

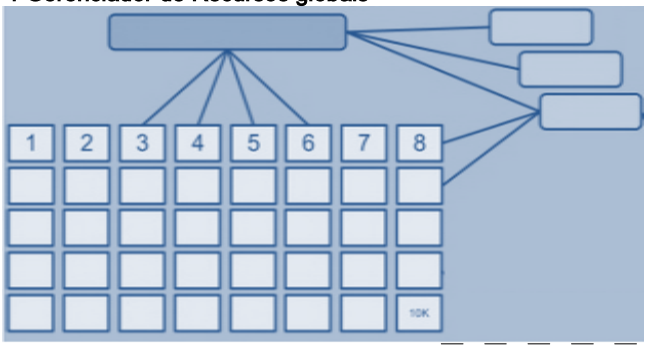
Arquitetura Hadoop 2.0

Características

- Desaparecimento do JobTracker

Manter o controle de recursos disponíveis (tarefas dos TaskTrackers) e monitorar os JobTrackers.

1 Gerenciador de Recursos globais



10 000 JobTracker - reduzidos

Um JobTrackers reduzido é criado para cada Job para monitorar suas tarefas(executa nos nós de trabalho).

10 000 JobTracker - TaskTracker

Ao invés de executar 40 TaskTrackers, executam 38 ou 39 e JobTracker reduzido.

Hadoop - Arquitetura do Hadoop 2.0: YARN

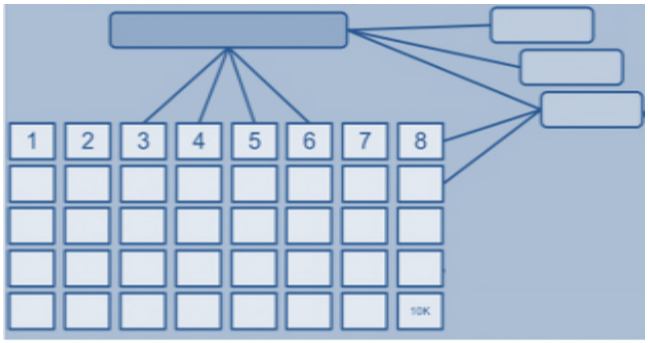
Arquitetura Hadoop 2.0

Características

- Novos nomes e funções no Hadoop 2.0

YARN -YET ANOTHER RESOURCE NEGOTIATOR

1 Gerenciador de Recursos globais – Resource Manager



10 000 Application Master

Um *Application Master* é criado para cada aplicação.

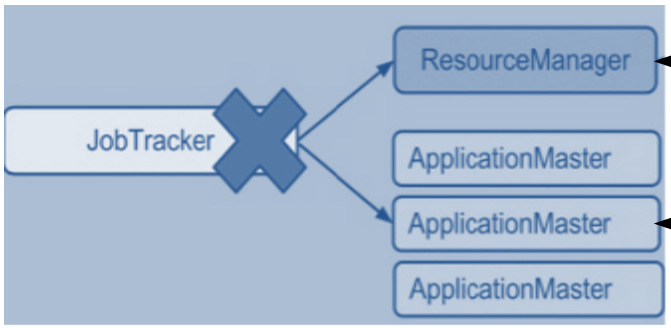
10 000 Node Manager

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Novos nomes e funções no Hadoop 2.0- YARN



Um para todo o cluster

Controle dos Application Masters

Um por aplicação

Monitoração e controle das tarefas

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Node Manager

  - Mais flexível e eficiente que o *TaskTracker*;

  - Executa qualquer tipo de computação que faz sentido no contexto do *Application Manager*: não somente *map e reduce*;

  - Possui o conceito de *containers:*

    - Podem lidar com recursos variáveis (RAM, CPU, IO);
    - Não exige número de funções map´s e reduce;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Node Manager

  Possui o conceito de *containers:*

  - Cria um *container* para cada tarefa
  - Possui propriedades físicas:

    1 CPU, 2GB RAM e 100GB disco;
  - Número de *containers* é limitado;
  - Não pode exceder os recursos do *Node Manager;*

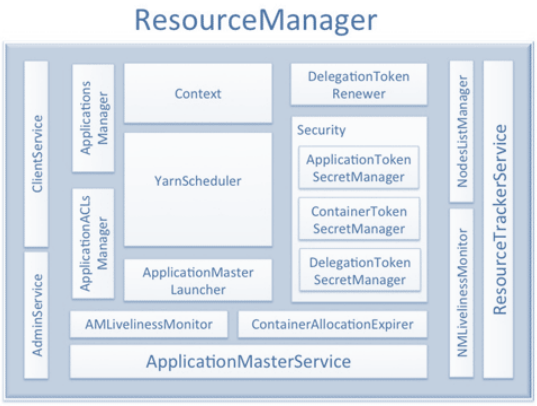
Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Características

- Resource Manager

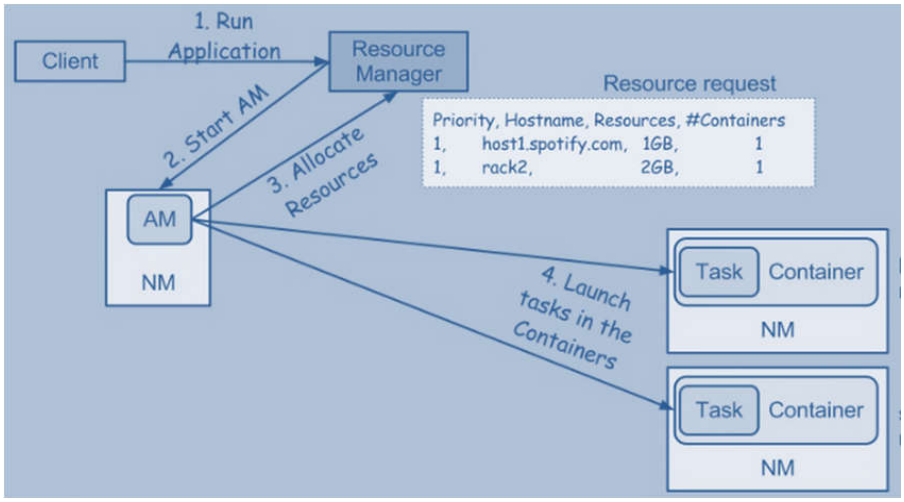
Na verdade é um pouco mais complexo:



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

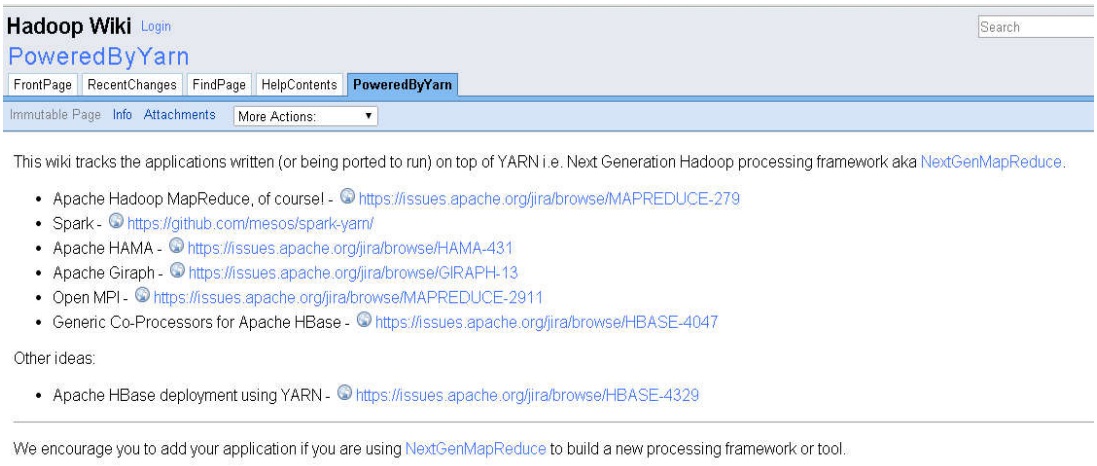
Requisição usando YARN



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Muitas novas aplicações usam YARN



Hadoop Wiki Login

Search

PoweredByYarn

FrontPage | RecentChanges | FindPage | HelpContents | PoweredByYarn

Immutable Page Info Attachments More Actions:

This wiki tracks the applications written (or being ported to run) on top of YARN i.e. Next Generation Hadoop processing framework aka NextGenMapReduce.

- Apache Hadoop MapReduce, of course! - https://issues.apache.org/jira/browse/MAPREDUCE-279
- Spark - https://github.com/mesos/spark-yarn/
- Apache HAMA - https://issues.apache.org/jira/browse/HAMA-431
- Apache Giraph - https://issues.apache.org/jira/browse/GIRAPH-13
- Open MPI - https://issues.apache.org/jira/browse/MAPREDUCE-2911
- Generic Co-Processors for Apache HBase - https://issues.apache.org/jira/browse/HBASE-4047

Other ideas:

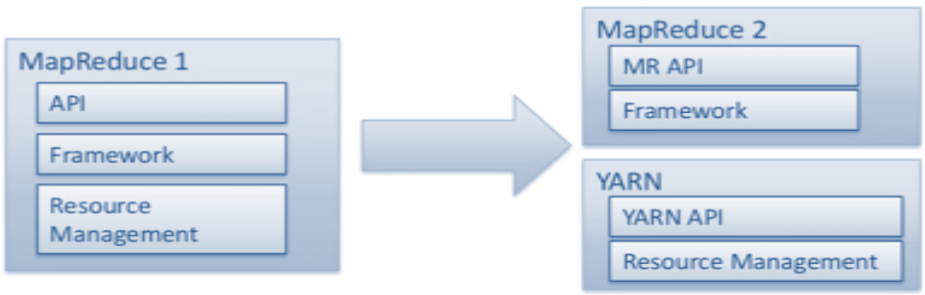
- Apache HBase deployment using YARN - https://issues.apache.org/jira/browse/HBASE-4329

We encourage you to add your application if you are using NextGenMapReduce to build a new processing framework or tool.

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

E o MapReduce???



Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

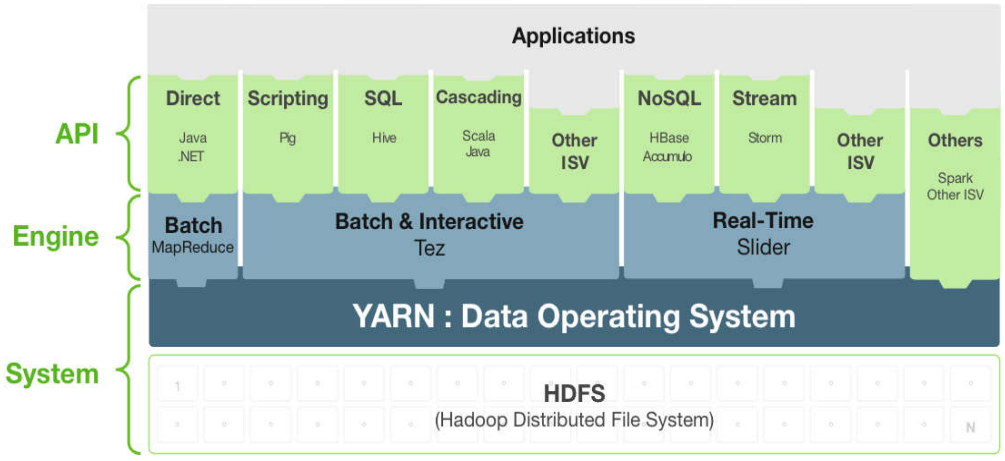
Na Prática....

- Hadoop se torna uma plataforma de processamento paralelo mais flexível;
- Não executar apenas MapReduce;
- Uma nova família de aplicações vem sendo desenvolvidas;
- Torna-se orientada a aplicações empresariais: flexibilidade;

Hadoop - Arquitetura do Hadoop 2.0: YARN

Arquitetura Hadoop 2.0

Na Prática....



Hadoop - Arquitetura do Hadoop 2.0: YARN

Atividade

- Exploração da interface do Ambari;
- Verificação de execução do YARN no Ambari;

Soluções para processamento paralelo e distribuído de dados

Ilustração de Oliver Munday

Soluções Processameto paralelo e distribuído de de dados

Spark - Básico

- Introdução ao Spark
- Histórico e motivações
- Arquitetura e Conceitos do Spark
- RDD´s
- Algumas Ações e transformações
- Spark SQL
-DataFrame API

Soluções para processamento paralelo e distribuído de dados

Spark – Introdução

Ilustração de Oliver Munday

Spark - Introdução

Definição:

- É um plataforma(em cluster) para processamento rápido, paralelo e de propósito geral;
- Extende o algoritmo padrão: MapReduce;
- Muitos cálculos são feitos em memória (podem ser executados no disco também);
- API´s: Java, Scala, Python, SQL e R

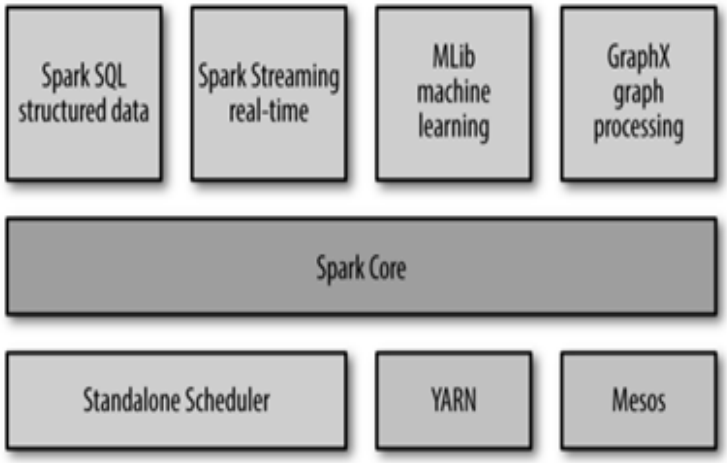
Spark - Introdução

Motivação:

- Ambiente único para execução de vários tipos de processamentos:
  - Batch
  - Processamentos interativas
  - Processamentos de streamings
- Muti-propósito: ML, Grafos, Processamento e Consultas;
- Stack de tecnologias

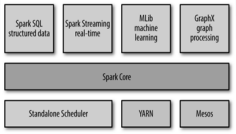
Spark - Introdução

Macro arquitetura:



Spark - Introdução

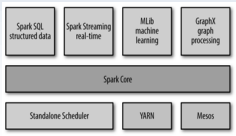
Spark Core:



- Funcionalidade básicas:
  - Agendamento de tarefas
  - Gerenciamento de memória
  - Tolerância a falhas
  - Interação com sistemas de armazenamento e outros...
- Possui a API dos RDD´s (*resilient distributed datasets*), principal abstração da programação no Spark;
- RDD´s = coleção de itens distribuídos no *cluster* e que podem ser manipulados de forma paralela;

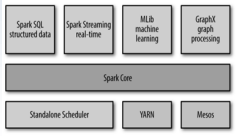
Spark - Introdução

Spark SQL:



- Componente para lidar com dados estruturados;
- Acesso a dados via SQL(de forma similar ao Hive do Hadoop);
- Suporta vários tipos de dados: Hive Tables, Parquet e arquivos Json;
- É possível num único programa mesclar SQL com outras operações suportadas pelos RDD´s;

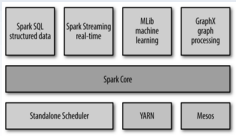
Spark - Introdução



Spark Streaming:

- Componente para lidar com *Streams* de dados;
- Desenvolvido, também,para lidar com logs de servidores ou filas de mensagens
- API do Spark Streaming, assim como as outras, são integradas ao API Spark Core;
- Fornece também: escalabilidade, tolerância a falhas, interações com dados em memória e disco;

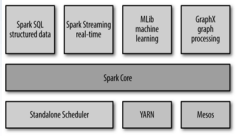
Spark - Introdução



Spark MLlib:

- Possui algoritmos típicos de aprenzidado de máquina;
- Alguns tipos: classificação, regressão, agrupamento, filtros colaborativos(recomendação);
- Parte do time que trabalhou no Mahout ajudou no desenvolvimento deste componente;
- Outras funcionalidades: importação de dados e avaliação de modelos;
- Oferece também suporte para algumas etapas comuns em alguns algoritmos(ex.: otimização via gradiente descendente)

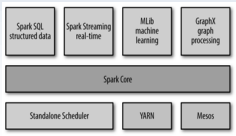
Spark - Introdução



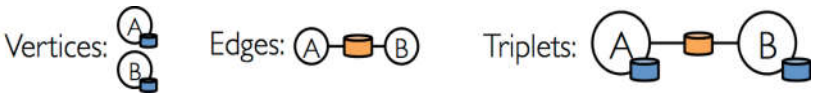
Spark GraphX:

- Componente para manipulação de grafos;
- Computação paralela é feita nos metódos de manipulação dos grafos;
- Exemplo de métodos: PageRank, detecção de triângulos, componentes conectados e outros

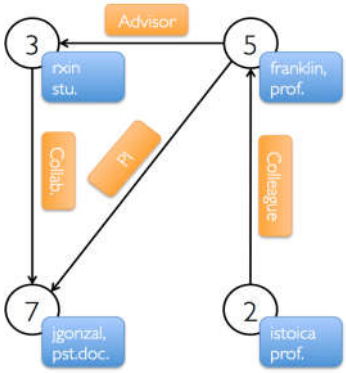
Spark - Introdução



Spark GraphX:



Property Graph



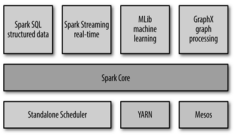
Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Spark - Introdução



Gerenciadores:

- Spark é escalável para milhares de nós em execução;

- Pode executar sobre vários gerenciadores de clusters;

-Exemplos: Hadoop YARN(integração), Apache Mesos ou mesmo uma versão simplificada do próprio Spark;

Soluções para processamento paralelo e distribuído de dados

Spark – Histórico e Motivações

Ilustração de Oliver Munday

Spark – Histórico e Motivações

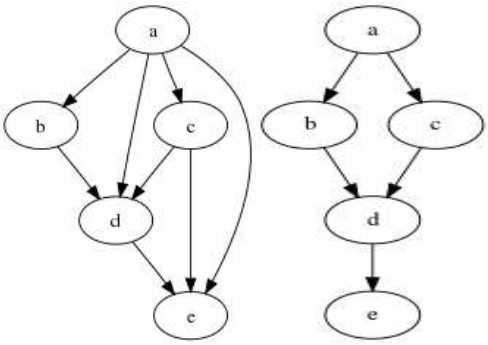
Histórico:

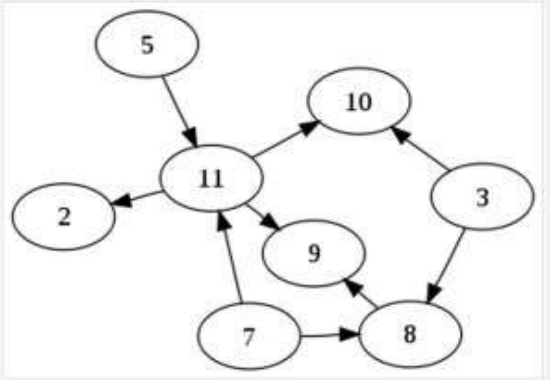
- É um projeto aberto e mantido por uma 'comunidade' de desenvolvedores;
- Projeto foi iniciado em 2009 na UC Berkeley RAD Lab;
  - Posteriormente é criada a AMPLab;
  - Boa parte do membros da laboratório trabalharam implementação MapReduce do Hadoop;
  - No Hadoop 1.0 perceberam, claramente, as limitações de MapReduce para jobs interativos e iterativos;
  - Em 2009 as primeiras versões do Spark são de 10 a 20 vezes mais rápidas que o MapReduce;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- DAG ou Grafos direcionados acíclicos



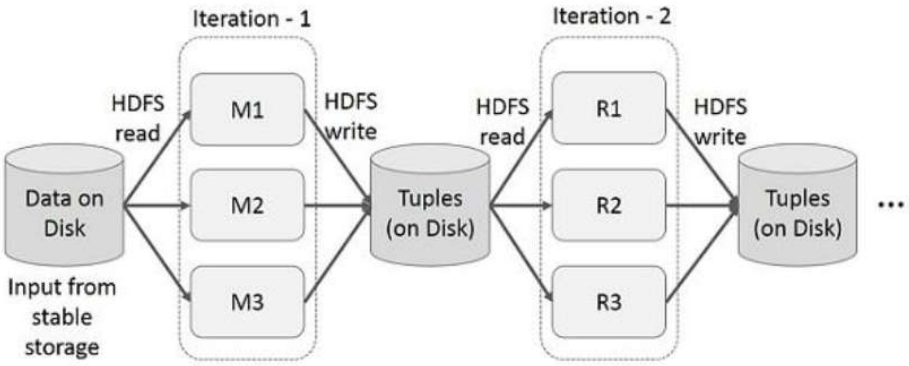


Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- MapReduce traduz as etapas da computação para apenas duas etapas;



Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Processos complexos tipicamente requerem vários etapas(por exemplo vários MapReduces 'aninhados');
- Na verdade estas etapas formam um DAG, que é uma generalização do MapReduce;
- Desta forma, DAG não conflita com o paradigma MapReduce;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Originalmente, MapReduce, tem dificuldades de paralelizar vários jobs MapReduce;
- A abordagem DAG resolve este problema;
- Imagine 3 Jobs: A, B e C:
  - C depende de A e B;
  - Então A e B podem executar de forma paralela (dois *maps,* por exemplo);
  - Esta precedência é explicitada em um DAG facilmente;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Conceitualmente a solução não podeira ser adaptada ao MapReduce?
- Caraterísticas MapReduce:
  - Ler dados do HDFS;
  - Processar resultados aplicando paradigma Map e Reduce;
  - Gravar dados novamente no HDFS;
  - Cada Job MapReduce é completamente independente um do outro;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Caraterísticas MapReduce:
  - O 'Name node' não faz ideia do que cada um dos MapReduce´s esta executando e não há possibilidade de 'aproveitamento' de dados;
  - Para lógicas iterativas estas características são totalmente indesejáveis;
  - Para processos interativos estas características também devem ser otimizadas;

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

DAG ou Grafos direcionados acíclicos

- Caraterísticas MapReduce:
  - Processo mais complexos exigem cadeias de processo MapReduce;
  - Com cadeias de processos MapReduce a paralelização é comprometida: o próximo Map só inicia após o seu Reduce predecessor;
  - Mesmo cadeias de processos com pequenos volumes de dados podem ter desempenho pobre:
    - Gravação em disco em todos as etapas;
    - Processo de inicialização para cada etapa;

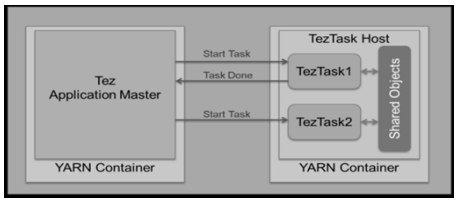
Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- Este é o problema resolvido pelo Spark, aliado com a utilização do dados reaproveitáveis e em memória;

- Outras implementações também fazem uso da mesma ideia, utilizando o YARN(Hadoop 2.X);

- Um exemplo é o TEZ que é utilizado, dentre outros, pela HortonWorks ;



Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- Proposta Spark :

  - Tratar todos os fluxos de processamento como DAG;

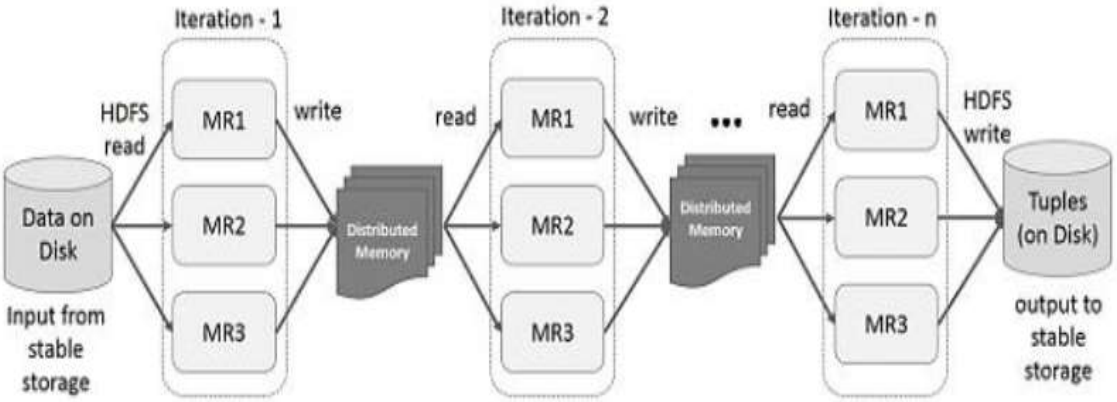
  - Carregar dados em memória e evitar leituras em disco em cada uma das etapas;

  - Utiliza os RDD´s para compartilhar dados e aproveitar para não fazer escrita no disco ao final de cada etapa (não escreve dados no HDFS ao fim de cada *Reduce*);

Spark – Histórico e Motivações

Histórico: Mas qual a solução adotada no Spark ?

- Proposta Spark :



Spark – Histórico e Motivações

Histórico: E depois ???

- Spark foi adotado por alguns grupos em Berkeley;
- Algumas empresas começaram a usar a solução também(Chaordic aqui no Brasil, por exemplo);
- Em 2010 o projeto teve seu código aberto;
- Em 2011 a AMPLab começou o desenvolvimento dos outros componentes Spark: Shark(Hive on Spark) e Spark Streaming;
- Em 2013 foi transferido para a fundação Apache

Soluções para processamento paralelo e distribuído de dados

Spark – Arquitetura e Conceitos do Spark

Ilustração de Oliver Munday

Spark - Arquitetura e Conceitos do Spark

Exemplo de programa Spark (*python*):

```
>>> lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07") #
Criação do RDD

>>> lines.count() # Operação no RDD

>>> lines.first() # linha 1 do arquivo
```

Conceitos importantes em uma aplicação Spark:

- *Driver Program*
- Contexto Spark
- *Executors*

Spark - Arquitetura e Conceitos do Spark

Driver Program

- Qualquer aplicação Spark é um '*Driver program*';
- Esta aplicação é capaz de fazer operações paralelas no cluster Spark;
- Possui um função principal (*main*) e:
    - Define/cria dados distribuídos no cluster;
    - Aplica operações paralelas aos dados distribuídos no cluster;
    - Operações não são apenas Map e Reduce;

Spark - Arquitetura e Conceitos do Spark

Contexto Spark

- Um 'Driver program' acessa o spark através de um objeto contexto;
- Na verdade, representa a conexão da aplicação com o Cluster;
- Usando o Spark Shell, automaticamente, um contexto é criado;

Spark - Arquitetura e Conceitos do Spark

Contexto Spark

- Para inicializar um contexto Spark em uma aplicação são necessários dois parâmetros:

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("My App")
sc = SparkContext(conf = conf)
```

- URL do Cluster : no exemplo acima usamos '*local* (executa o Spark em uma única *thread* na máquina local sem usar o cluster);

- Nome da aplicação: irá identificar o nome de sua aplicação no gerenciador do *cluster* ;

- Outros métodos;

Spark - Arquitetura e Conceitos do Spark

Contexto Spark - SparkConf

Instance Methods		
	`__init__(self, loadDefaults=True, _jvm=None)` Create a new Spark configuration.	source code
	`set(self, key, value)` Set a configuration property.	source code
	`setMaster(self, value)` Set master URL to connect to.	source code
	`setAppName(self, value)` Set application name.	source code
	`setSparkHome(self, value)` Set path where Spark is installed on worker nodes.	source code
	`setExecutorEnv(self, key=None, value=None, pairs=None)` Set an environment variable to be passed to executors.	source code
	`setAll(self, pairs)` Set multiple parameters, passed as a list of key-value pairs.	source code
	`get(self, key, defaultValue=None)` Get the configured value for some key, or return a default otherwise.	source code
	`getAll(self)` Get all values as a list of key-value pairs.	source code
	`contains(self, key)` Does this configuration contain a given key?	source code
	`toDebugString(self)` Returns a printable version of the configuration, as a list of key=value pairs, one per line.	source code
Inherited from object: `__delattr__, __format__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __sizeof__, __str__, __subclasshook__`		

Spark - Arquitetura e Conceitos do Spark

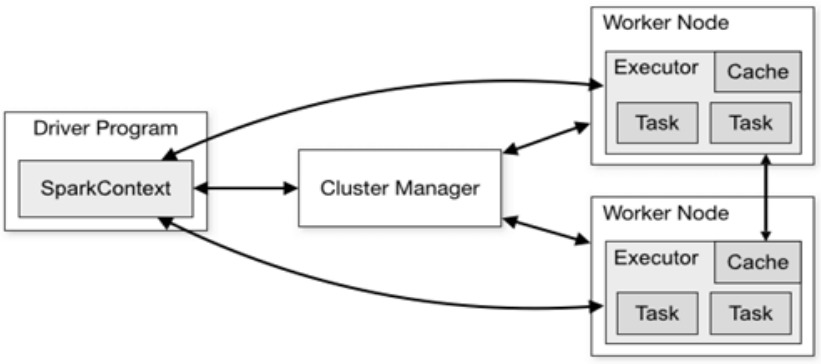
Contexto Spark - SparkContext

Instance Methods		
	`__init__(self, master=None, appName=None, sparkHome=None, pyFiles=None, environment=None, batchSize=1024, serializer=PickleSerializer(), conf=None)` Create a new SparkContext.	source code
	`defaultParallelism(self)` Default level of parallelism to use when not given by user (e.g.	source code
	`__del__(self)`	source code
	`stop(self)` Shut down the SparkContext.	source code
	`parallelize(self, c, numSlices=None)` Distribute a local Python collection to form an RDD.	source code
	`textFile(self, name, minSplits=None)` Read a text file from HDFS, a local file system (available on all nodes), or any Hadoop-supported file system URI, and return it as an RDD of Strings.	source code
	`union(self, rdds)` Build the union of a list of RDDs.	source code
	`broadcast(self, value)` Broadcast a read-only variable to the cluster, returning a `Broadcast` object for reading it in distributed functions.	source code
	`accumulator(self, value, accum_param=None)` Create an `Accumulator` with the given initial value, using a given `AccumulatorParam` helper object to define how to add values of the data type if provided.	source code
	`addFile(self, path)` Add a file to be downloaded with this Spark job on every node.	source code
	`clearFiles(self)` Clear the job's list of files added by `addFile` or `addPyFile` so that they do not get downloaded to any new nodes.	source code
	`addPyFile(self, path)` Add a .py or .zip dependency for all tasks to be executed on this SparkContext in the future.	source code
	`setCheckpointDir(self, dirName)` Set the directory under which RDDs are going to be checkpointed.	source code
Inherited from object: `__delattr__, __format__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __sizeof__, __str__, __subclasshook__`		

Spark - Arquitetura e Conceitos do Spark

Executor

- Para realizar as operações do '*Driver Program*', o mesmo faz uso de vários nós que possuem '*Executor*':



Spark - Arquitetura e Conceitos do Spark

Exemplo de Wordcount - Scala

```scala
// Create a Scala Spark Context.
val conf = new SparkConf().setAppName("wordCount")
val sc = new SparkContext(conf)
// Load our input data.
val input =  sc.textFile(inputFile)
// Split it up into words.
val words = input.flatMap(line => line.split(" "))
// Transform into pairs and count.
val counts = words.map(word => (word, 1)).reduceByKey{case (x, y) => x + y}
// Save the word count back out to a text file, causing evaluation.
counts.saveAsTextFile(outputFile)
```

Spark - Arquitetura e Conceitos do Spark

Prática – Interagindo com o Spark Shell Python

Prática – Interagindo com o Spark Shell Scala

Soluções para processamento paralelo e distribuído de dados

Spark – RDD´s

Ilustração de Oliver Munday

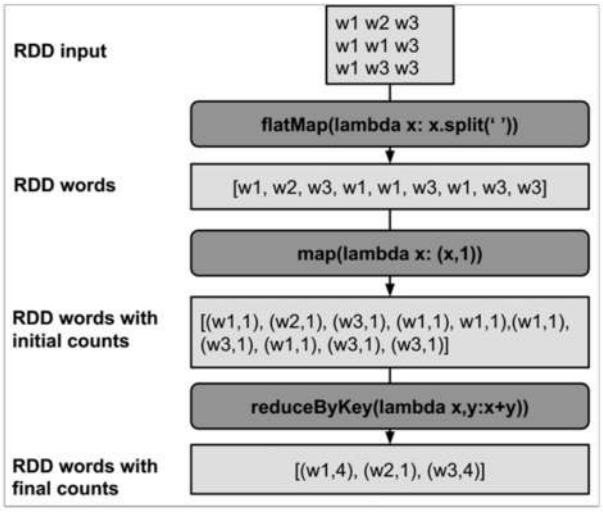
Spark - RDD´s

Conceitos

- É uma coleção distribuída de elementos;
- Lógica de trabalho no Spark:
  - Criar RDD´s
  - Transformar RDD´s existentes
  - Executar operações nos RDD´s para gerar resultados
- De forma transparente:
  - Spark distribui os dados dos RDD´s no cluster;
  - Paraleliza as operações executadas nos RDD´s

Spark - RDD´s

Conceitos



Spark - RDD´s

Conceitos

- RDD´s representam um conjunto distribuídos de objetos 'imutáveis';
- Imutável = não volatilidade:
  - Facilita o processo de distribuição, replicação e compartilhamento dos dados
  - Dados imutáveis podem permanecer em disco ou memória de maneira indistinta
- Facilitam a tolerância a falhas

Spark - RDD´s

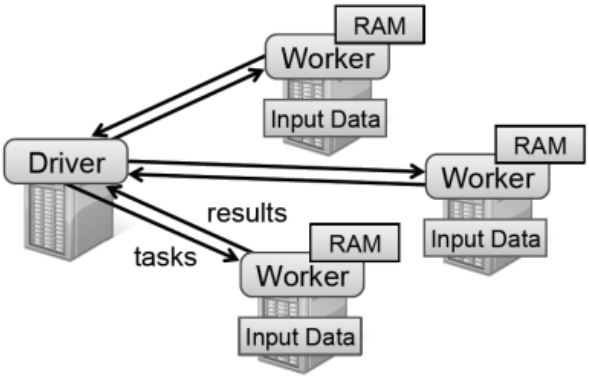
Conceitos

- RDD´s são divididos em várias '*partitions*';
- As '*partitions*' podem residir em vários nós do *Cluster*;
- Criação de RDD´s é feito de duas maneiras
  - Carregando dados de forma externa, por exemplo;
  - Distribuindo uma coleção de objetos no programa *Driver*
- O *Driver program* é quem coordena as atividades e os *Workers* armazenam e manipulam as partições dos RDD´s

Spark - RDD´s

Conceitos

- Driver:
  - Define e invoca os ações(actions) nos RDD´s
  - Rastreia as alterações nos RDD´s
- Workers:
  -Armazenam as partições dos RDD´s
  - Executam transformações (transformations) nos RDD´s



Spark - RDD´s

Conceitos

- Uma vez criado os RDD´s duas operações são possíveis;
- Spark trata estas duas operações de maneira distinta;
- **Transformations**:
  - Cria um RDD a partir de um anterior;
  - Exemplo é o *transformation filter*

```
lines.filter(lambda line: "Python" in line)
```

  - O resultado do comando anterior irá gerar outro RDD, resultado do filtro aplicado;
  - Sempre retornam um RDD

Spark - RDD´s

Conceitos

- **Actions**
  - Calculam resultados a partir de um RDD;
  - Este resultado é então enviado para o *Driver program* ou pode ser salvo, por exemplo, no HDFS;
  - Retornam algum outro tipo de dados, com exceção de RDD;
  - Os valores de retorno são enviados para o *Driver program* ou persistidos em disco;
  - Por padrão todas as vezes que um *action* é acionada os RDD´s envolvidos são recalculados;

Spark - RDD´s

Conceitos

- **Actions**;
  - Forçam a execução dos *transformations;*
  - Exemplos de ações já vistas anteriormente:

```
>>> lines.count() # Operação no RDD
>>> lines.first() # linha 1 do arquivo
```

Spark - RDD´s

Conceitos

- *Lazy Evaluation*
  - As transformações (*transformations*) são sempre postergadas até encontrar uma ação (*action*);
  - A principio isto pode parecer estranho, mas para Big Data faz todo o sentido;
  - Deve se considerar que todo programa Spark é um DAG, por conceito de implementação. Veja o programa abaixo:

```
lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07")
lines.count()
```

Spark - RDD´s

Conceitos

- *Lazy Evaluation*

  - Considere este programa agora:

```
lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07")
lines.first()
```

  - O conceito de avaliação tardia em conjunto com os DAG´s fazem com as otimizações de código no Spark sejam muito boas, conforme o exemplo anterior;

  - OBS.: por padrão os RDD´s são recalculados a cada vez que uma '*action*' é executada;

Spark - RDD´s

Conceitos

- *Lazy Evaluation*

  - Este conceito altera o entendimento sobre RDD´s:

    - São formas de calcular *stages* de processamento;

    - Formas como as lógicas são realizadas a partir do dado de origem;

    - Um programa spark, em essência, segue a lógica de um processo ETL (clássico em ambientes de DW);

    - Uma recomendação importante é: quebre seu programa spark, via de regra, em várias estágios de processamento;

Spark - RDD´s

Conceitos

- Noções de persistência

  - Para reusar o RDD em várias '*actions*' o comando *'persist'* deve ser explicitado

  - Há algumas opções para persistência dos dados;

  - Uma vez que o *'persist'* é utilizado, não haverá mais recálculos quando houver outras '*actions*';

  - Pragmaticamente: você vai persistir os dados em memória uma vez e fazer várias consultas ('*actions*') nestes dados em em memória;

Spark - RDD´s

Conceitos

- Noções de persistência

  Exemplo:

```
>>> lines = sc.textFile("/apps/hive/warehouse/sample_07/sample_07") # Criação do RDD
>>> lines.persist() # persistência dos dados ou cache() para memoria

>>> lines.persist().is_cached # Esta na memoria

>>> lines.count() # Operação no RDD
>>> lines.first() # linha 1 do arquivo
```

Spark - RDD´s

Conceitos

- Noções de persistência

StorageLevel	Espaço usado	CPU	Em memória	Em Disco
MEMORY_ONLY	Alto	Baixo	S	N
MEMORY_ONLY_SER	Baixo	Alto	S	N
MEMORY_AND_DISK	Alto	Médio	Parcial	Parcial
MEMORY_AND_DISK_SER	Baixo	Alto	Parcial	Parcial
DISK_ONLY	Baixo	Alto	N	S

- Há o método *unpersist()*;

Spark - RDD´s

Prática – Verificando processos e RDD´s