Apache Spark

Contents

1. Introduction to Spark

- Overview of Spark
- Spark is fast
- Spark is ubiquitous
- Additional key features of Spark
- Common uses of Spark

Overview of Spark

- Spark was created as an alternative to the Hadoop MapReduce Framework
  - Spark has a rich API for developing big data apps
  - Spark code is also much more concise than MapReduce (e.g. 1:5 code quantity)

- MapReduce has only 2 data processing operations - map and reduce
  - You have to break every problem down into a sequence of map and reduce jobs - this is hard!

- Spark has approx. 100 data processing operations
  - Much richer and expressive API, better suited to complex tasks

Spark is Fast

- Spark is orders of magnitude faster than MapReduce
  - This is a critical factor for businesses relying on timely information
- Reason #1 - Spark uses in-memory cluster computing
  - MapReduce reads/writes data to disk
  - Spark allows an app to cache data in memory for processing
  - Reading data in memory is 100 times faster than reading from disk
- Reason #2 - Spark has an advanced job execution engine
  - MapReduce requires complex data processing algorithms to be split into multiple sequential jobs, which prevents MapReduce from doing any optimization
  - Spark doesn't force the developer to split it into multiple sequential jobs, which means Spark can do optimizations

Spark is Ubiquitous

- Spark is general-purpose - it has an integrated set of libraries to do the following types of data processing jobs:
  - Batch processing
  - Stream processing
  - Interactive analysis
  - Machine learning
  - Graph computing

- Benefits of using Spark:
  - No need to learn multiple frameworks or to copy data from one silo system to another

Additional Key Features of Spark

- Spark is scalable
  - To increase data processing capacity of a Spark cluster, just add more nodes
  - You can start with a small cluster and grow as necessary
  - No code changes required when you add a node to a Spark cluster
- Spark and Hadoop are fault tolerant
  - Spark automatically handles node failures
  - No need for developers to handle these failures in your code

Common Uses of Spark

- Iterative algorithms
  - i.e. data processing algorithms that iterate over the same data multiple times
  - e.g. apps that run hundreds of iterations of some algorithm over the same data
  - Iterative algorithms run fast on Spark, due to its in-memory computing capabilities (i.e. apps can cache data in memory)

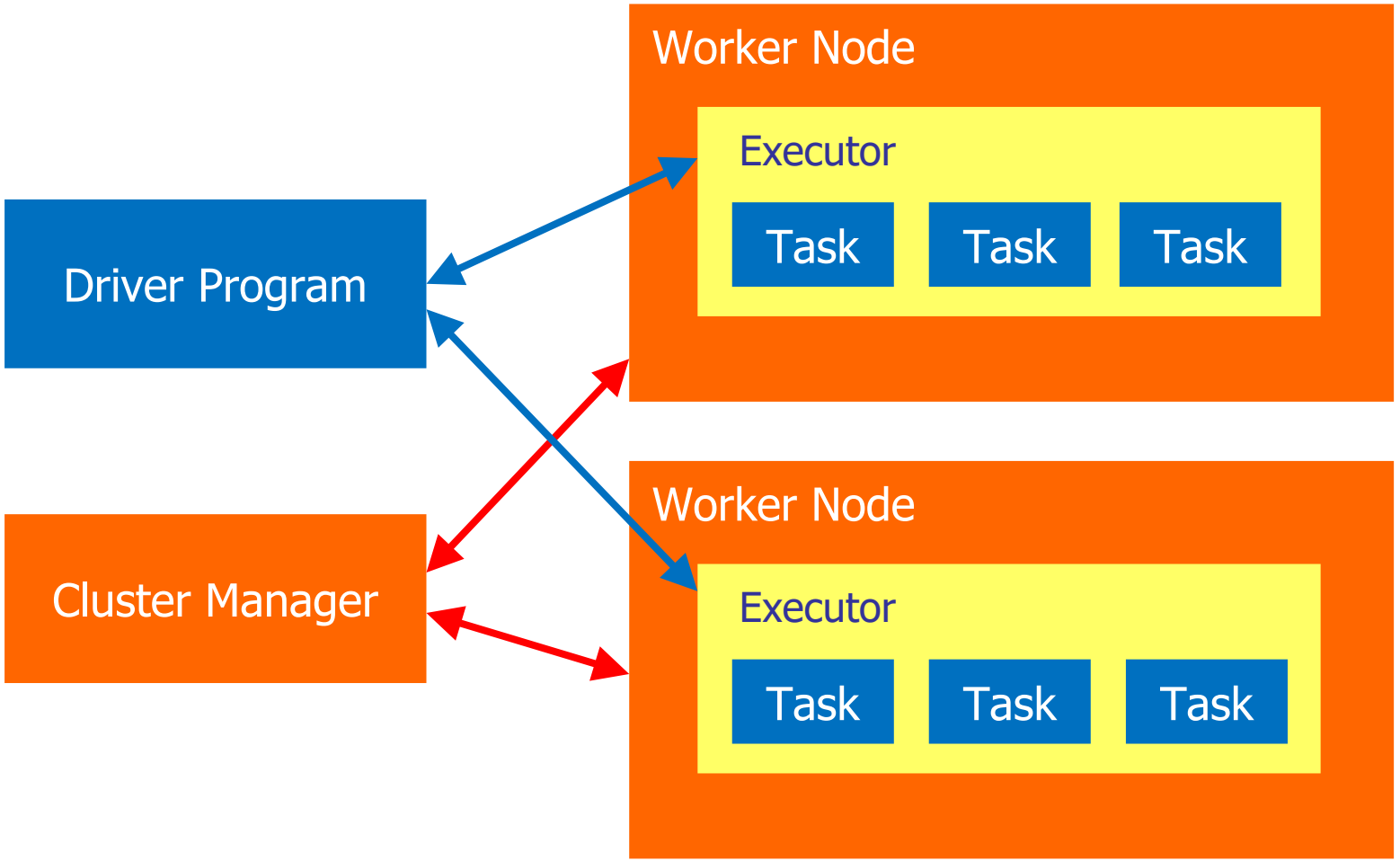
- Interactive analysis
  - i.e. exploring a dataset interactively
  - e.g. do a summary analysis of a large dataset before running a batch processing job that might take hours
  - Spark is well-suited for interactive analysis, again due to its in-memory computing capabilities (caching avoids multiple disk hits)

2. Spark Architecture

- Overview
- Worker nodes and cluster managers
- Driver programs, executors, and tasks

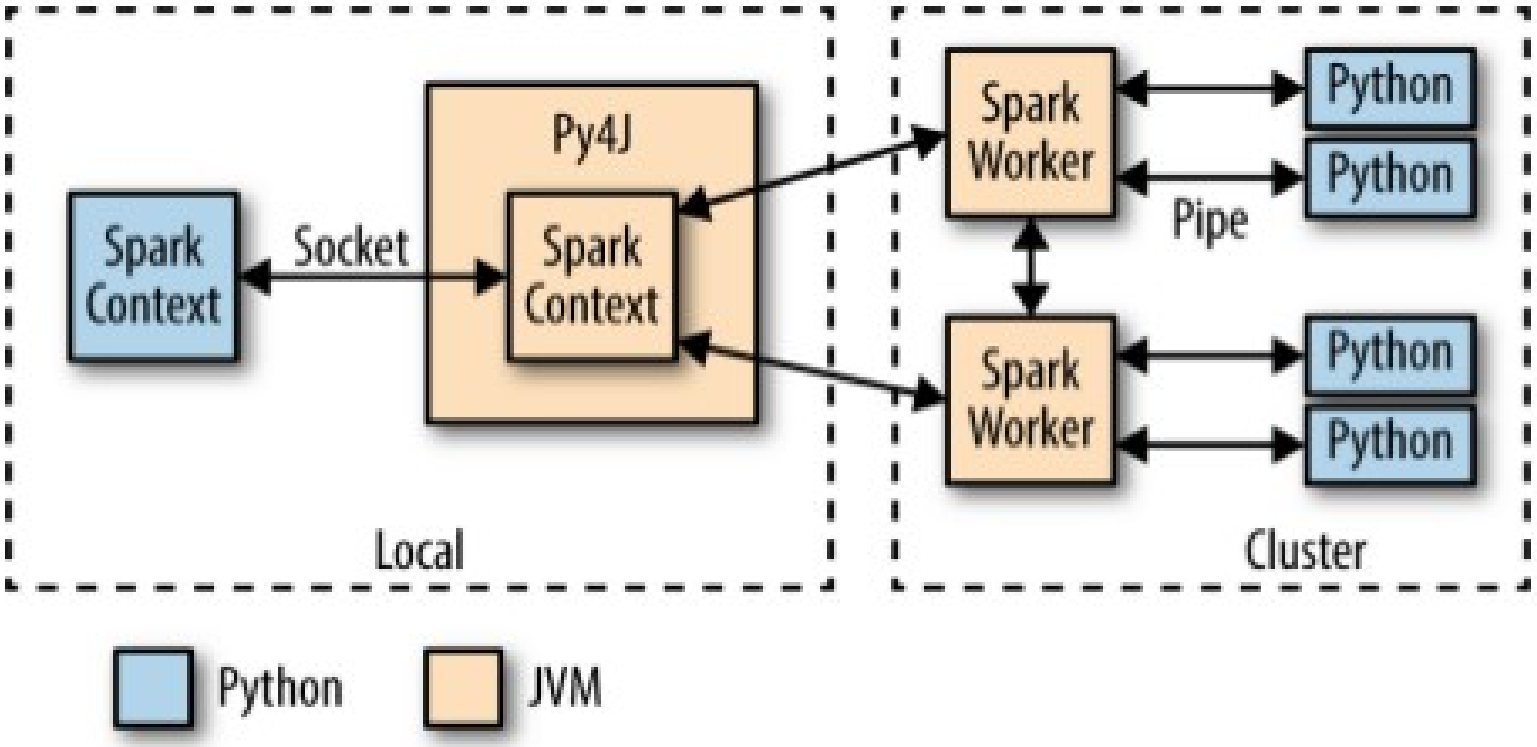
Overview

- Here's the high-level architecture of Spark



PySpark

- The Python interface to spark is very popular, called 'pyspark'
- Spark is written in Java, PySpark uses a library called Py4J to convert between python and Java



Worker Nodes and Cluster Managers

- Worker nodes
  - A worker node provides CPU, memory, and storage to a Spark app
  - Worker nodes run a Spark app as distributed processes on a cluster of nodes

- Cluster managers
  - A cluster manager manages computer resources across a cluster of worker nodes
  - Provides low-level scheduling of cluster resources across apps
  - Enables multiple apps to share cluster resources and run on the same worker nodes
  - Spark supports 3 cluster managers: Standalone, Mesos, and YARN

Driver Programs, Executors, and Tasks

- Driver programs
  - A driver program is an application that uses the Spark library
  - Has data processing code for Spark to execute on worker nodes
  - Can launch one or more jobs on a Spark cluster

- Executors
  - An executor is a JVM process that Spark creates on each worker node for an application
  - Executes application code concurrently on multiple threads
  - Can also cache data in memory or on disk

- Tasks (i.e. threads)
  - A task is the smallest unit of work that Spark sends to an executor
  - Executed by a thread in an executor on a worker node

3. Application Execution

- Overview
- Jobs
- Shuffles
- Stages
- How does a Spark application operate?
- Where does the data live?

Overview

- In this section we'll explain how applications work in Spark
  - i.e. how does a Spark application process data in parallel across a cluster of nodes

- Before we dive in, there's some important Spark terminology we need to explain first:
  - Shuffles
  - Jobs
  - Stages

Jobs

- What is a job?
  - Technically, a job is when you invoke an action operation on an RDD - see later for details
  - Spark will execute the work across the cluster of nodes - the degree of parallelism depends on what operations you perform
  - A job returns results to a driver program

- An application can launch multiple jobs
  - i.e. it can call several action operations on RDDs

Shuffles

- What is a shuffle:
  - A shuffle redistributes data among a cluster of nodes
  - This is an expensive operation, because it involves moving data across a network

Stages

- What is a stage?
  - Spark splits a job into a directed acyclic graph (DAG) of stages
  - A stage is a collection of operations
  - A stage may depend on another stage - e.g. a job may be split into two stages, where stage #2 can't begin until stage #1 is complete

- Spark groups tasks into stages using shuffle boundaries
  - Tasks that don’t need a shuffle are grouped into the same stage
  - A task that requires its input data to be shuffled begins a new stage

How does a Spark Application Operate?

- When a Spark app is run...
  - Spark connects to a cluster manager, and acquires executors on the worker nodes
  - Spark splits a job into a DAG of stages
  - Spark then schedules execution of these stages on the executors
  - The executors run tasks submitted by Spark, in parallel

- Every app has its own set of executors on worker nodes
  - Tasks from different apps isolated from each other - separate JVMs
  - An errant task in one app can't crash another app

- Disadvantage of apps running in separate JVMs...
  - Apps can't easily share data, unless they write to disk (expensive)
  - Apps sharing data through disk will experience performance issues

Where does the Data Come From?

- Spark isn't a data storage system - it works in conjunction with external storage systems, such as:
  - HDFS
  - HBase
  - Cassandra
  - MongoDB
  - Local file system
  - Spark SQL

- Spark can work with any Hadoop-compatible data source
  - Hadoop compatibility is important for organizations
  - E.g. can easily switch from using Hadoop MapReduce to Spark

4. Using the Spark Shell

- Overview
- Downloading the Spark shell
- Installing and configuring the Spark shell
- Running the Spark shell
- REPL commands
- Running system commands
- Entering Scala expressions

Overview

- Spark provides a rich API for processing rich data in a distributed architecture
  - We'll explore the details in the coming chapters

- An easy way to get started with Spark is to use the Spark shell
  - A command-line REPL (read-evaluate-print-loop) tool
  - Based on the Scala shell
  - Allows you to use Spark interactively in Scala

Any Questions?