Spark Overview

2015

Professor Sasu Tarkoma

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

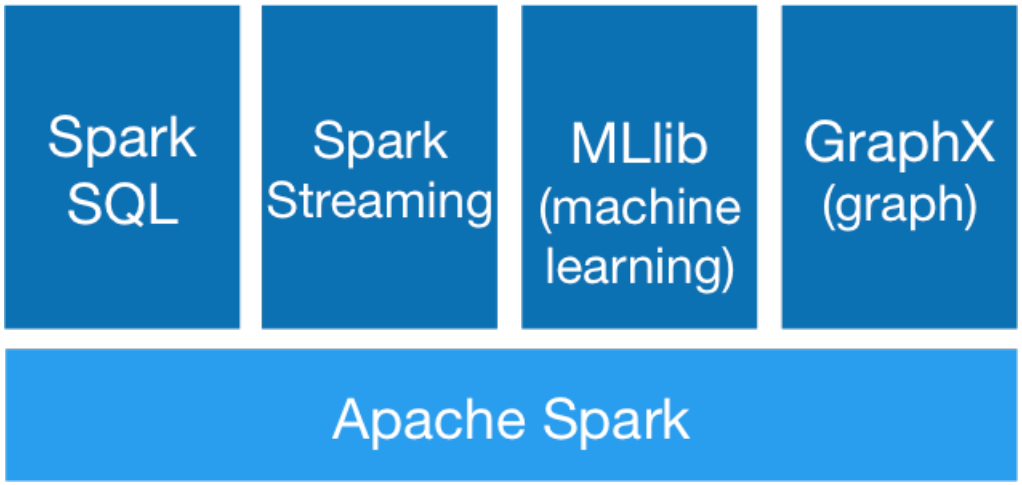
Apache Spark

- Spark is a general-purpose computing framework for iterative tasks
- API is provided for Java, Scala and Python
- The model is based on MapReduce enhanced with new operations and an engine that supports execution graphs
- Tools include Spark SQL, MLLlib for machine learning, GraphX for graph processing and Spark Streaming

Spark Aim

Unifies batch, streaming, interactive computing

Making it easy to build sophisticated applications



Key idea in Spark

Resilient distributed datasets (RDDs)

- Immutable collections of objects across a cluster
- Built with parallel transformations (map, filter, …)
- Automatically rebuilt when failure is detected
- Allow persistence to be controlled (in-memory operation)

Transformations on RDDs

- Lazy operations to build RDDs from other RDDs
- Always creates a new RDD

Actions on RDDs

- Count, collect, save

Spark terminology I/II

Application is a user program built on Spark that consists of a *driver program* and *executors* on the cluster.

Driver program is the process running the main() function of the application and creating the SparkContext

Cluster manager is an external service for acquiring resources on the cluster

Worker node is any node that can run application code in the cluster

Executor is a process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors

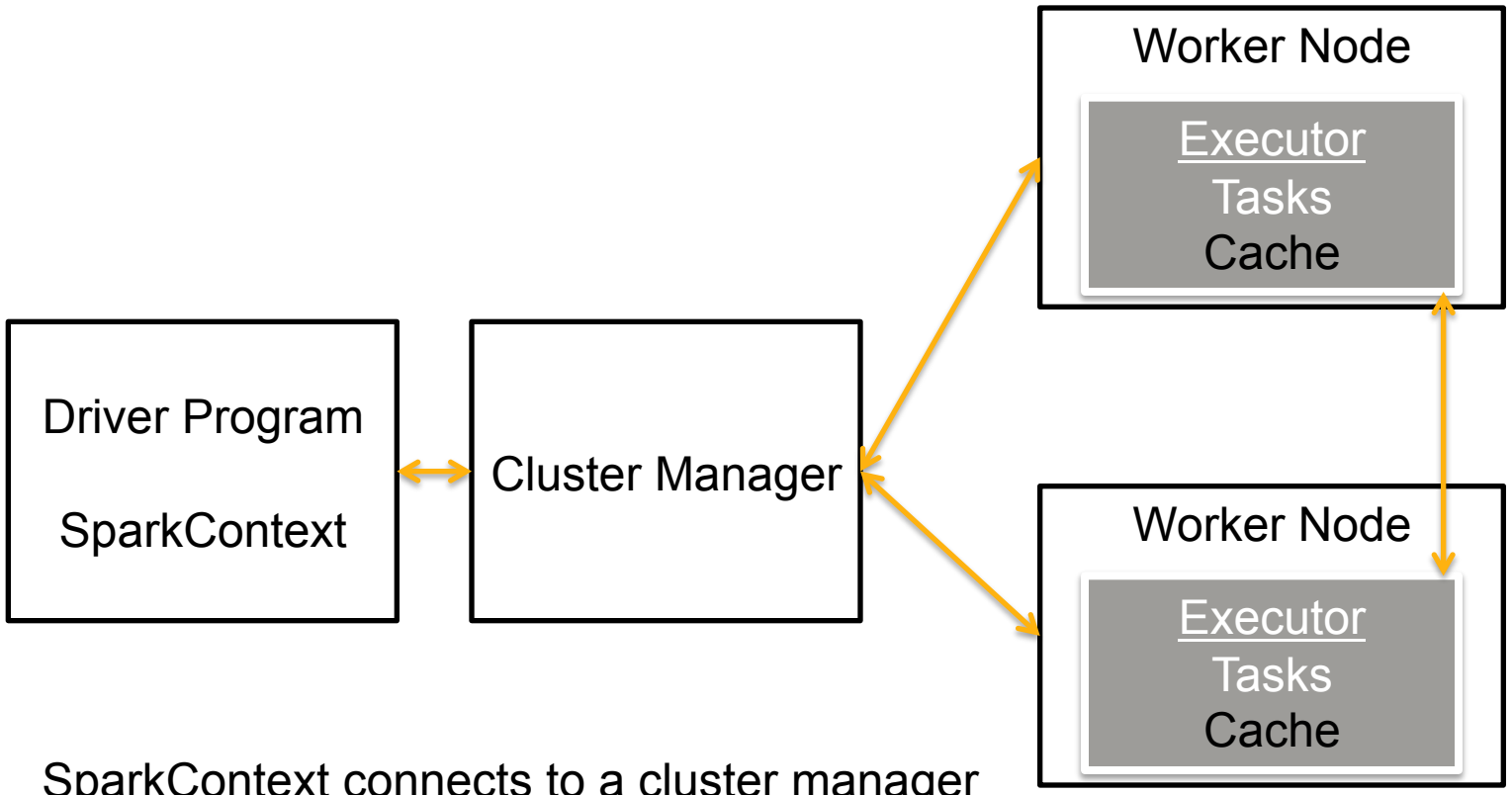
Spark terminology II

Task is a unit of work that will be sent to one executor

Job is a parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. save, collect)

Stage is about each job being divided into smaller sets of tasks called *stages* that depend on each other (similar to the map and reduce stages in MapReduce)

Spark overview



SparkContext connects to a cluster manager
Obtains executors on cluster nodes
Sends app code to them
Sends task to the executors

Spark cluster components

The SparkContext object in the main program controls the Spark applications

They are executed as independent sets of processes on a cluster

SparkContext can connect to several types of cluster managers (Spark native or Mesos/YARN) that are responsible for allocating resources across the cluster

Three observations on the cluster architecture

1. Applications are isolated from each other: tasks from different apps run in different JVMs. As a consequence data cannot be shared across Spark applications without using external storage.
2. Spark is flexible regarding the cluster manager. It must be able to acquire executor processes that communicate with each other.
3. The driver schedules tasks on the cluster it should reside near the worker nodes.

Cluster manager types

Standalone

- Simple cluster manager

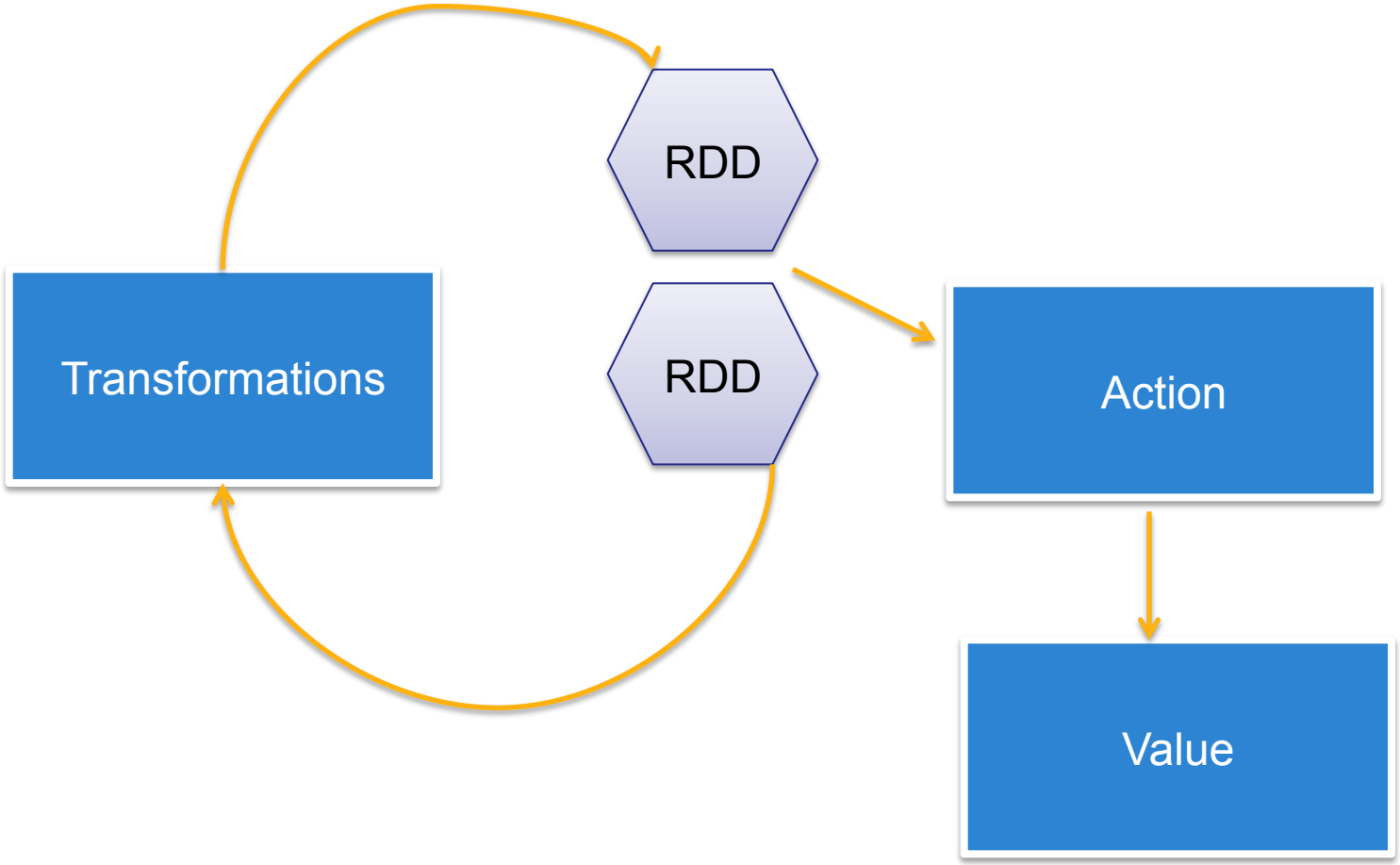
Apache Mesos

- A general cluster manager

Hadoop YARN

- the resource manager of Hadoop 2

How Spark works



Resilient Distributed Datasets (RDD)

RDDs document the sequence of transformations that were used to create them (the lineage)

This can be used to recompute lost data

RDD Partitions

Partitions support the management of parallel collections

One task is run for each partition of the cluster

Typically 2-4 partitions are used for each CPU

Spark sets this automatically, but it can be also manually tuned
(e.g. sc.parallelize(data, 10)).

Also the term slices is used for partitions

RDD I/II

RDD is the primary abstraction

Fault-tolerant collection of elements

Intrinsic support for parallel operations

Two different types of RDD:

- Parallelized collections based on Scala collections
- Hadoop datasets that allow the execution of functions on each record of a file in HDFS or some other storage system supported by Hadoop

RDD II

Two types of operations are supported by Spark

Transformations

- Lazy

Actions

- Operate on RDDs
- Transformed RDD is recomputed when action is applied

RDD can be stored into memory or disk

Persistence in Spark

Spark can cache (persist) RDD in memory

Each node stores in memory any parts of an RDD that it computes

A node can reuse these cached results when implementing new actions on the RDD, this can result in 10x performance compared to Hadoop

The node cache has fault tolerance so that if any partition of an RDD is lost, the lost part can be recomputed using the history of transformations

Persistence in Spark II

Spark monitors cache usage of each node and old data partitions are removed using an LRU (Least-Recently-Used) scheme

It is also possible to manually remove an RDD with the RDD.unpersist() method

Spark Persistence

Transformation	Description
MEMORY_ONLY	RDD is stored as deserialized Java objects in the JVM. If the RDD does not fit into the memory, it will be only partially cached and missing partitions will be recomputed when they are required. This is the default mode.
MEMORY_AND_DISK	The RDD is stored as deserialized Java objects in the JVM. If the RDD does not fit into memory, it will be partially accessed from disk.
MEMORY_ONLY_SER	The RDD is stored as serialized Java objects (one by array for a partition). This approach is more byte-efficient than the former methods; however, requires more CPU to read. Recomputes missing partitions when required.
MEMORY_AND_DISK_SER	Similar to the above case, but uses disk instead of recomputing the missing partitions.
DISK_ONLY	Store the RDD only to disk.
MEMORY_ONLY_2…	Same as above, but with replication to two clusters.

Shared Variables

A function given to a Spark operation is typically executed on a remote cluster node (for example map and reduce)

The function works on separate copies of the data and variables that are not shared by the remote cluster nodes

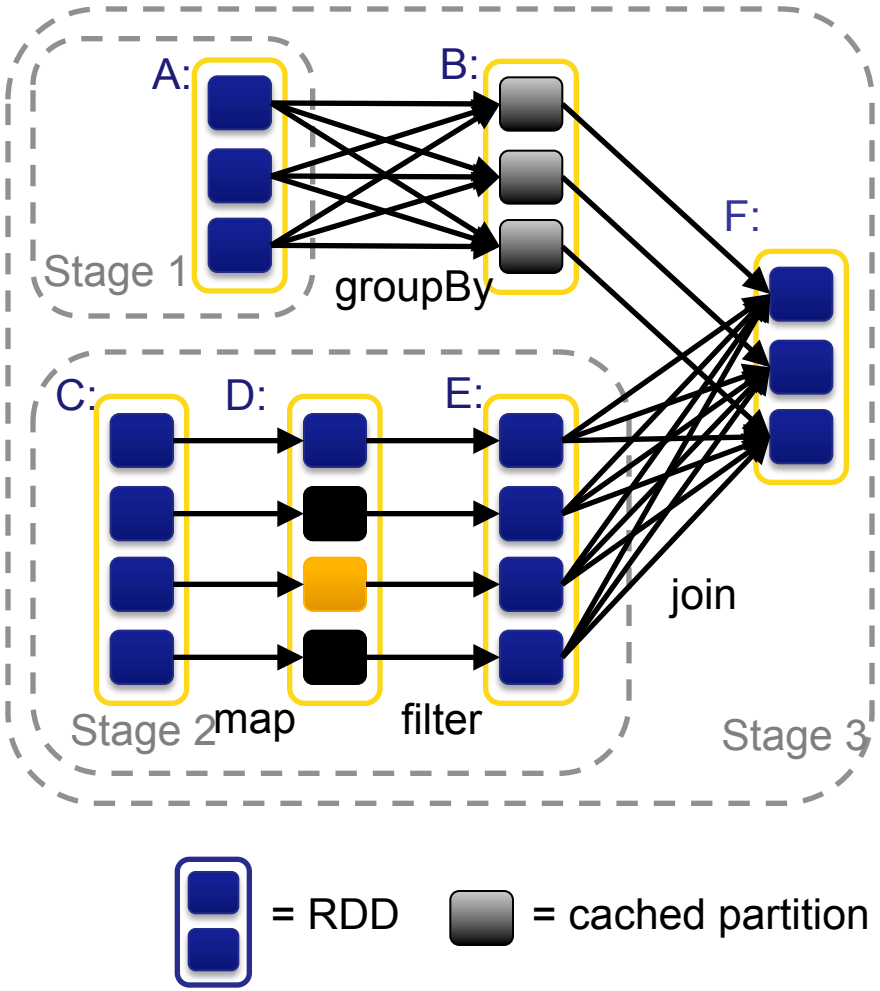
The necessary data and variables are copied to the remote node and no updates are sent back to the driver program

Spark supports two methods for shared variables:

- broadcast variables
- accumulators

Task Scheduler

- Supports general task graphs
- Pipelines functions where possible
- Cache-aware data reuse & locality
- Partitioning-aware to avoid shuffles



M.Zaharia. Parallel programming with Spark. O'Reilly Strata Conference. February 2013.