Spark @ Goldschmiede

von Reinis für Goldschmiede

Abstract

Für mich ist Apache Spark zu einem „Schweizer Taschenmesser“ der Datenverarbeitung geworden. Nicht nur für BigData oder Machine Learning Aufgaben, sondern für viele ETL-Aufgaben. Sowohl in Batch wie auch in Echtzeit.

In der Einführung dieser Goldschmiede stelle ich einige Anwendungsbereiche von Spark vor und gehe kurz auf das Map-Reduce Paradigma und die RDDs ein.

Danach tauchen wir gemeinsam in die Architektur von Spark ein und schauen uns einige wichtige Design-Elemente dieses Frameworks detaillierter an.

In der Produktion wird Apache Spark in einer Cluster-Umgebung ausgeführt. Wir werfen einen Blick auf die Ausführung auf dem Mesos anhand eines Produktivsystems und sprechen eventuell kurz die Ausführung in einer Hadoop-Umgebung durch (leider habe ich keine Demo für Hadoop).

Bevor wir uns selbst in einigen Beispielen von der Mächtigkeit von Spark überzeugen können, mache ich meinen subjektiven und durchaus befangenen Vergleich zwischen Spark @ Mesos und Hadoop.

Werkzeuge: Bitte bringt Eure Laptops mit. Eine IDE ist zwar nicht zwingend erforderlich, ich werde jedoch ein SBT-Beispielprojekt in den kommenden Tagen in der Goldschmiede-Git-Repo pushen. Wenn Ihr aber lieber mit CLI arbeitet, reicht es wenn Ihr die Spark-Distribution unter http://spark.apache.org/downloads.html herunterladet (v1.6.1 pre-built vor Hadoop 2.6 and later).

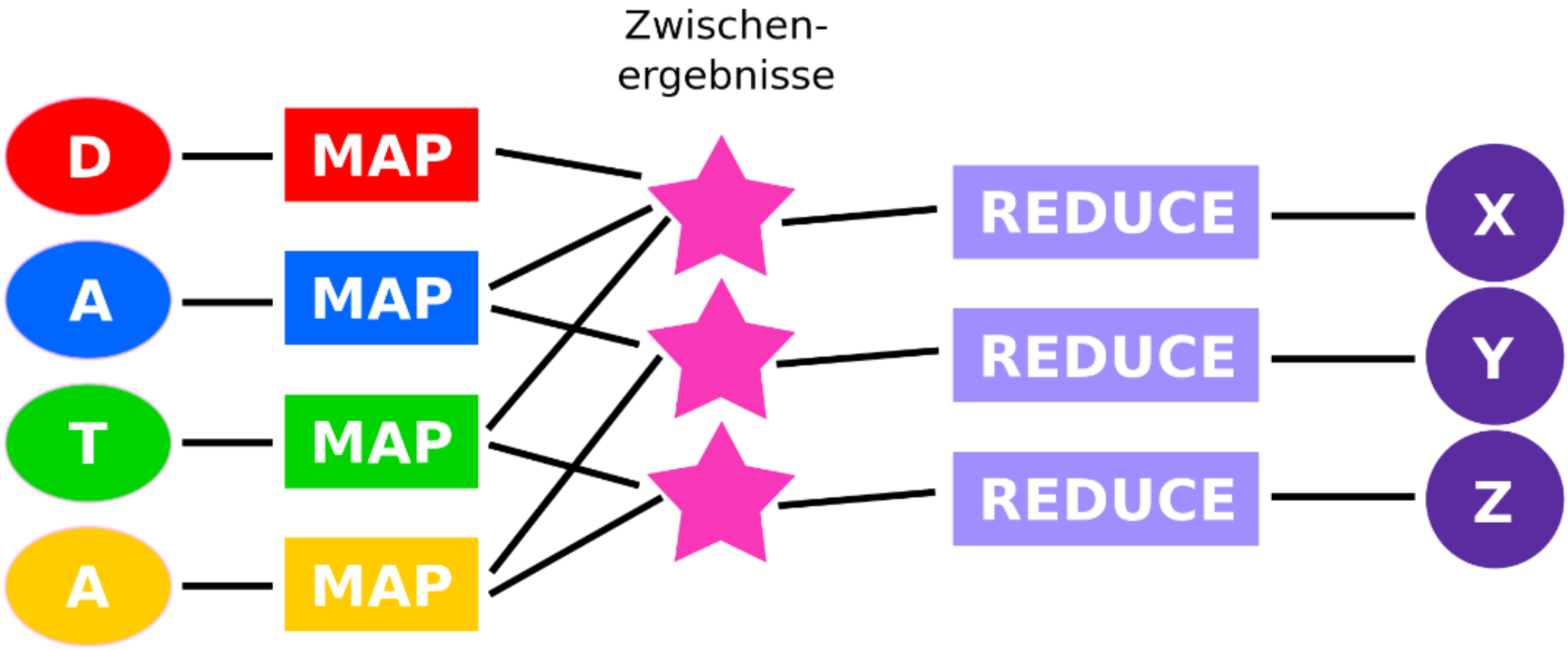
Inhalte

- **Einführung**
  - Anwendungsbereiche von Spark
  - Was ist Map-Reduce
  - Was sind RDDs
- Apache Spark im Detail
  - Spark Komponenten
  - Architekturübersicht
- Ausführen auf einem Cluster
  - Mesos
  - Standalone
  - Hadoop
- Spark vs Hadoop
  - Popularität Spark (+ Mesos) vs Hadoop
  - API Komplexität
  - Performance
  - Skalierbarkeit
- Beispiele

Anwendungsbereiche von Spark

- Aufgaben, welche mit SQL schwierig sind
  - Benutzerdefinierte SerDe Aufgaben
  - Batch mit Transformationen über Bibliotheken der Drittanbieter
  - Transformation zu einer de-normalisierten Datenstruktur
  - „Sparse“-Datenstrukturen bearbeiten
- Wenig zu sehr Viel zu Wenig
  - Transformationen mit großen Zwischenprodukten
- Sehr Viel zu sehr Viel
  - Massive Datenmenge lesen, Transformation durchführen, massive Datenmenge schreiben
- Batch zusammen mit Echtzeit-Verarbeitung
  - Gleiche oder ähnliche Transformationen im Batch und Echtzeit
- Parallel lesen/schreiben
  - Abhängig von Architektur von Quelle / Ziel

Was ist Map-Reduce



Quelle: https://de.wikipedia.org/wiki/MapReduce

Was sind RDDs

- Widerstandsfähig (R) Verteilt (D) Datasets (D)
  - Unveränderbar (Immutable)
  - Partitioniert
  - Fehlertolerant
  - Erzeugt durch grobgranulare Befehle (coarse-grained operations)
  - Verzögerte Auswertung (lazy evaluation)
  - Können gespeichert werden
  - …

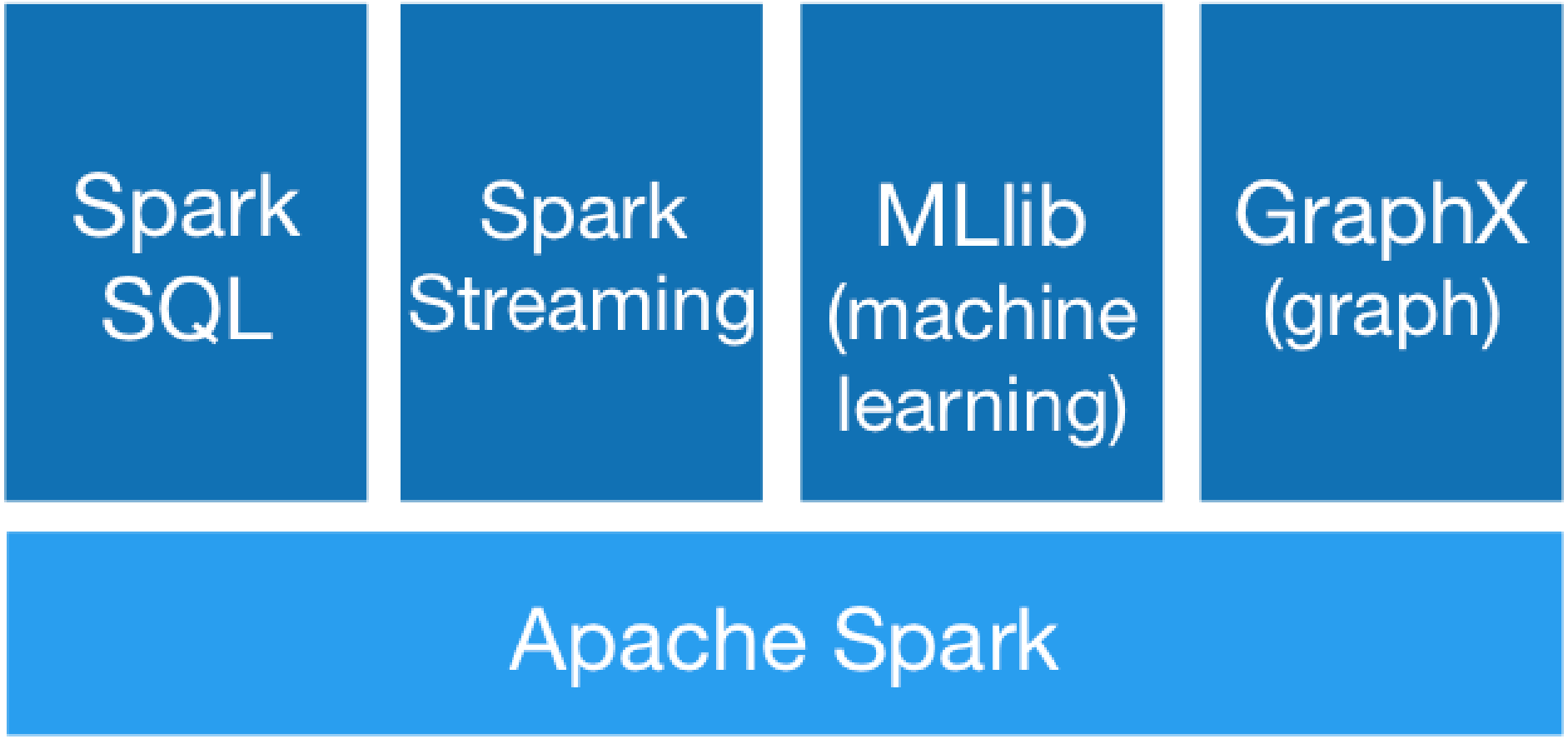
Speicherung

```
import org.apache.hadoop.hbase.client.Put
import org.apache.hadoop.hbase.io.ImmutableBytesWritable
import org.apache.hadoop.hbase.mapreduce.TableOutputFormat
import org.apache.hadoop.conf.Configuration
import org.apache.hadoop.hbase.HBaseConfiguration

trait HBasePuttable { def put: Put }

case class Document (id: Int, content: String, mimeType: String, creationDate: DateTime) extends HBasePuttable {
  def put = {
    import my.HBaseConversions._
    new Put(id)
      .add(HBaseModel.DEFAULT_CF, HBaseModel.Document.documentId, documentId)
      .add(HBaseModel.DEFAULT_CF, HBaseModel.Document.content, content)
      .add(HBaseModel.DEFAULT_CF, HBaseModel.Document.mimeType, mimeType)
      .add(HBaseModel.DEFAULT_CF, HBaseModel.Document.creationDate, creationDate.getMillis)
  }
}

def saveToHTable[T <: HBasePuttable](context: SparkContext, rdd: RDD[T], tableName: String): Unit = {
    val hConf = HBaseConfiguration.create()
    hConf.set(TableOutputFormat.OUTPUT_TABLE, "/path/to/hbase/tables/table")
    hConf.setClass(MRJobConfig.OUTPUT_FORMAT_CLASS_ATTR, classOf[TableOutputFormat[ImmutableBytesWritable]], classOf[OutputFormat[ImmutableBytesWritable, Put]])

    val immutableWritable = context.broadcast(new ImmutableBytesWritable())

    rdd.map { t => immutableWritable.value -> t.put }.saveAsNewAPIHadoopDataset(hConf)
  }
}
```

Inhalte

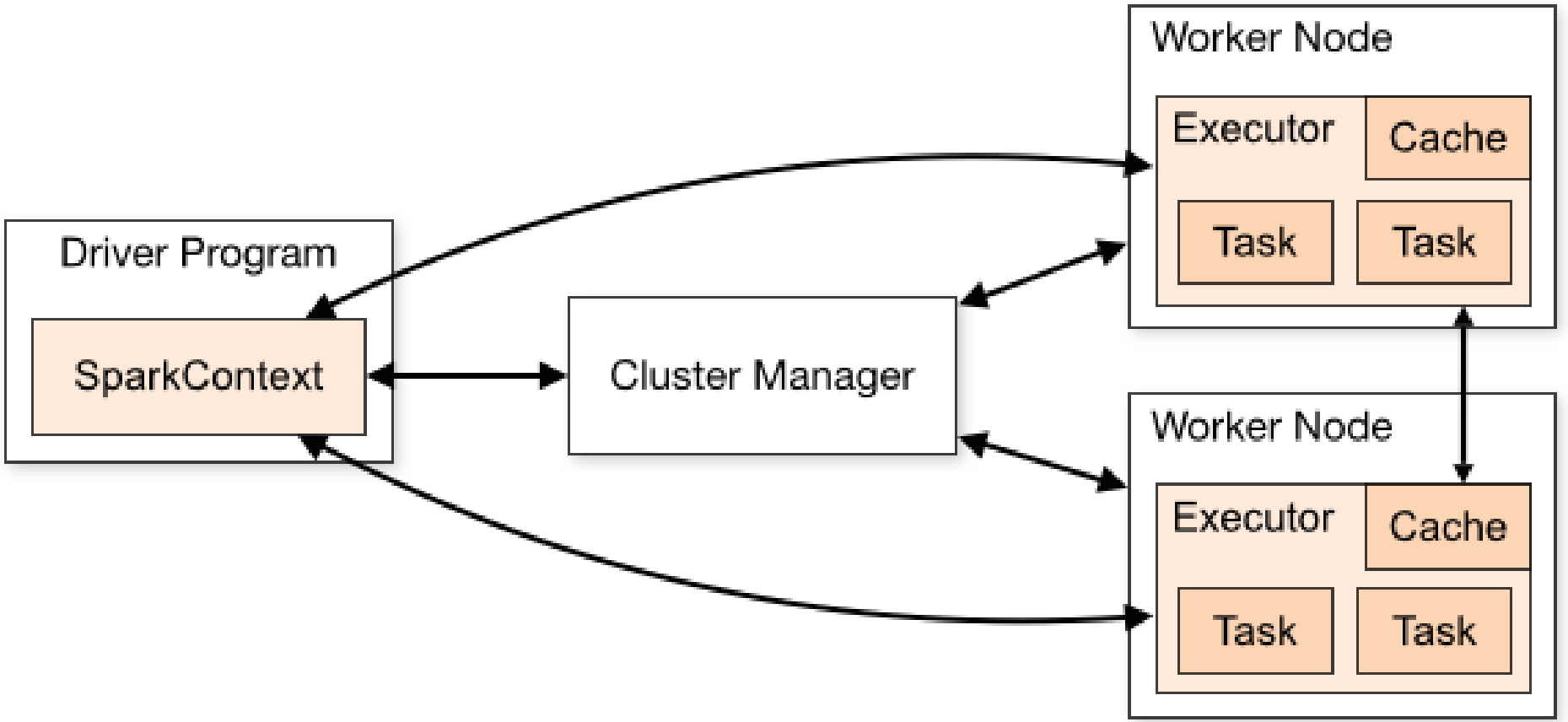
- Einführung
  - Anwendungsbereiche von Spark
  - Was ist Map-Reduce
  - Was sind RDDs
- **Apache Spark im Detail**
  - Spark Komponenten
  - Architekturübersicht
- Ausführen auf einem Cluster
  - Mesos
  - Standalone
  - Hadoop
- Spark vs Hadoop
  - Popularität Spark (+ Mesos) vs Hadoop
  - API Komplexität
  - Performance
  - Skalierbarkeit
- Beispiele

Spark Komponenten



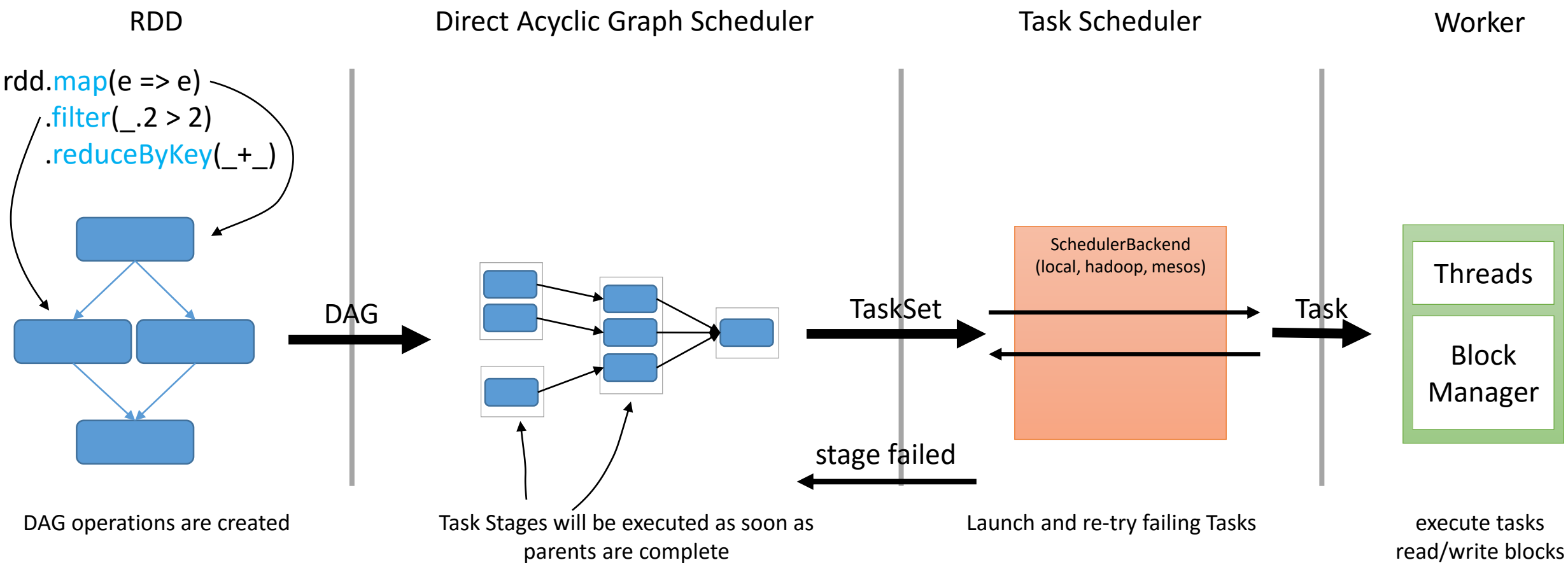
Quelle: https://spark.apache.org/

Architekturübersicht

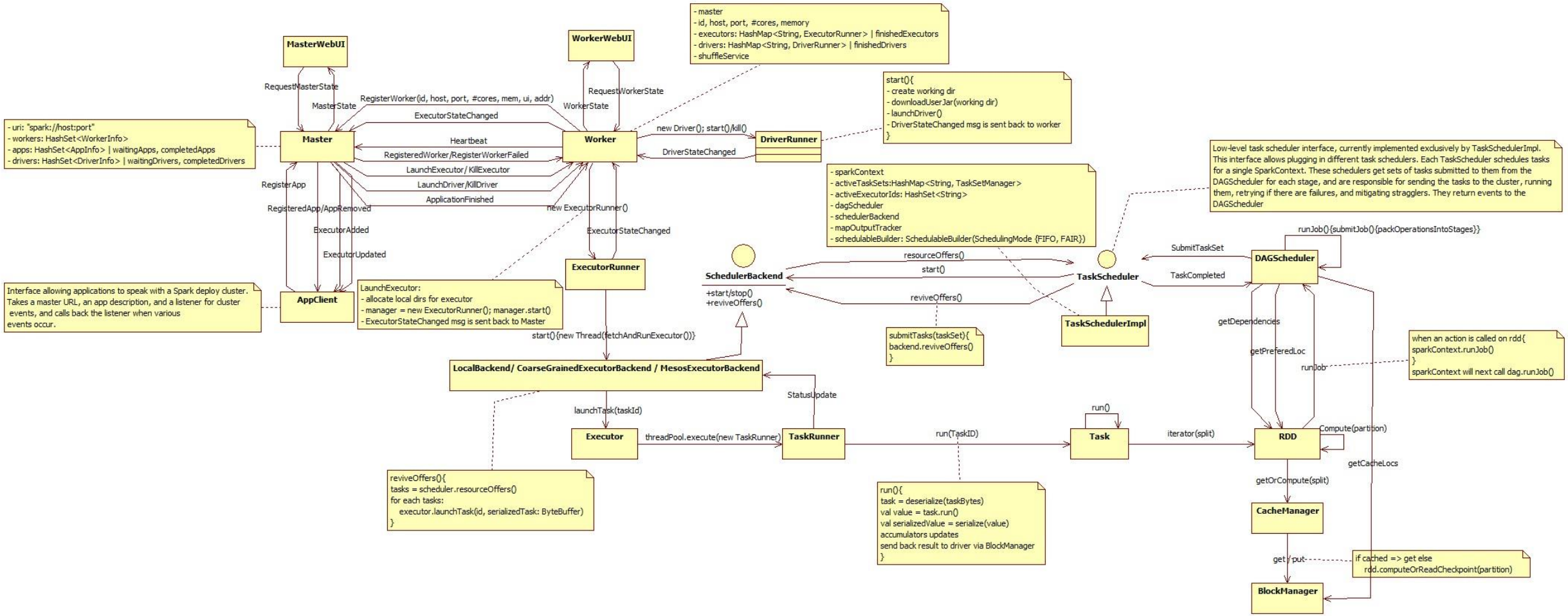


Quelle: https://spark.apache.org/docs/1.6.1/cluster-overview.html

Architekturübersicht



Architekturübersicht*



* Quelle: http://www.trongkhoanguyen.com/2015/03/apache-spark-13-architecture-module.html

Inhalte

- Einführung
  - Anwendungsbereiche von Spark
  - Was ist Map-Reduce
  - Was sind RDDs
- Apache Spark im Detail
  - Spark Komponenten
  - Architekturübersicht
- **Ausführen auf einem Cluster**
  - Standalone
  - Mesos
  - Hadoop
- Spark vs Hadoop
  - Popularität Spark (+ Mesos) vs Hadoop
  - API Komplexität
  - Performance
  - Skalierbarkeit
- Beispiele

Ausführen auf einem stand-alone Cluster

./sbin/start-master.sh

./sbin/start-slave.sh <master-spark-URL>

conf/spark-env.sh.template

./bin/spark-shell --master spark://IP:PORT

http://spark.apache.org/docs/latest/spark-standalone.html

Ausführen auf einem Mesos Cluster

Ausführen auf einem Hadoop Cluster

```
spark-submit --master yarn --deploy-mode cluster \
--class „HadoopBeispiel" hdfs:///test-assembly-1.0-SNAPSHOT.jar
```

Inhalte

- Einführung
  - Anwendungsbereiche von Spark
  - Was ist Map-Reduce
  - Was sind RDDs
- Apache Spark im Detail
  - Spark Komponenten
  - Architekturübersicht
- Ausführen auf einem Cluster
  - Mesos
  - Standalone
  - Hadoop
- **Spark vs Hadoop**
  - Popularität Spark (+ Mesos) vs Hadoop
  - API Komplexität
  - Performance
  - Skalierbarkeit
- Beispiele

Popularität* Spark (+ Mesos) vs Hadoop

Spark (+ Mesos)

- Spark: Excluding merges, 105 authors have pushed 484 commits to master and 525 commits to all branches. On master, 1,591 files have changed and there have been 59,194 additions and 40,436 deletions.
- Mesos: Excluding merges, 55 authors have pushed 439 commits to master and 490 commits to all branches. On master, 520 files have changed and there have been 35,315 additions and 16,458 deletions.

Hadoop

- Excluding merges, 44 authors have pushed 169 commits to trunk and 528 commits to all branches. On trunk, 757 files have changed and there have been 23,291 additions and 9,627 deletions.

* Git Pulse vom 11. April 2016

API Komplexität

```
public static void main(String[] args) throws Exception {
    String inputPath=args[0];
    String outputPath=args[1];
    JobConf conf = new JobConf(SampleMapReduce.class);
    conf.setJobName("SampleMapReduce");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(SampleMapper.class);
    conf.setCombinerClass(SampleReducer.class);
    conf.setReducerClass(SampleReducer.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(inputPath));
    FileOutputFormat.setOutputPath(conf, new Path(outputPath));

    JobClient.runJob(conf);
}
```

```
@Stringable
@InterfaceAudience.Public
@InterfaceStability.Stable
public class Text
extends BinaryComparable
implements WritableComparable<BinaryComparable>
```

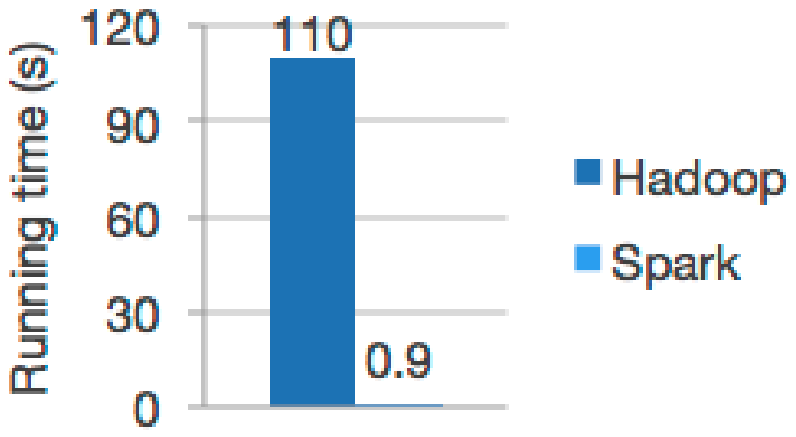
```
public static class SampleMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable count = new IntWritable(1);
  private Text text = new Text();
  public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {
    String line = value.toString ();
    StringTokenizer tokenizer = new StringTokenizer (line);
    while (tokenizer.hasMoreTokens()) {
      text.set(tokenizer.nextToken());
      Output.collect(text, count);
    }
  }
}
```

```
public static class SampleReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }
```

Performance

Spark ist schneller weil

- Hadoop I/O auf die Festplatte bei jedem Map und Reduce
- Cluster muss oft getuned werden für spezifischen Job
- Spark cached in-memory
- Spark boot-strap viel schneller



Cluster tuning

STEP 2: Worker Host Planning

Now that you have your base Host configuration from Step 1, use the table below to allocate resources, mainly CPU and memory, to the various software components that run on the host.

Service	Category	CPU (cores)	Memory (MB)	CM Static Service %	Notes
Operating System	Overhead	1	8192	N/A	Most operating systems use 4-8GB minimum.
Cloudera Manager agent	Overhead	1	1024	N/A	Allocate 1GB for Cloudera Manager agents, which track resource usage on a
Other services	Overhead	0	0	N/A	Enter the required cores or memory for services not listed above.
HDFS DataNode	CDH	1	1024	4	Allocate 1GB for the HDFS DataNode.
Impala daemon	CDH	0	0	0	(Optional Service) Suggestion: Allocate at least 16GB memory when using Im
Hbase RegionServer	CDH	0	0	0	(Optional Service) Suggestion: Allocate no more than 12-16GB memory when

STEP 4: YARN Configuration on Cluster

These are the first set of configuration values for your cluster. You can set these values in YARN->Configuration in Cloudera Manager.

YARN Configuration Property	Value	
yarn.nodemanager.resource.cpu-vcores	176	Copied from STEP 2 "Available Resources"
yarn.nodemanager.resource.memory-mb	250880	Copied from STEP 2 "Available Resources"

STEP 5: Verify YARN Settings on Cluster

Go to the Resource Manager Web UI (usually http://<ResourceManagerIP>:8088/ and verify the "Memory Total" and "Vcores Total" matches the values above. If your machine has no bad nodes, then the numbers should match exactly.

Resource Manager Property to Check	Value	Note
Expected Value for "Vcores Total"	1760	Calculated from STEP 2 "YARN Available Vcores" and STEP 3
Expected Value for "Memory Total" (in GB)	2450	Calculated from STEP 2 "YARN Available Memory" and STEP 3

STEP 6: Verify Container Settings on Cluster

In order to have YARN jobs run cleanly, you need to configure the container properties.

YARN Container Configuration Property (Vcores)	Value	Description
yarn.scheduler.minimum-allocation-vcores	1	Minimum vcore reservation for a container
yarn.scheduler.maximum-allocation-vcores	32	Maximum vcore reservation for a container
yarn.scheduler.increment-allocation-vcores	1	Vcore allocations must be a multiple of this value

YARN Container Configuration Property (Memory)	Value	
yarn.scheduler.minimum-allocation-mb	1024	Minimum memory reservation
yarn.scheduler.maximum-allocation-mb	8192	Maximum memory reservation
yarn.scheduler.increment-allocation-mb	512	Memory allocations must be a

STEP 6B: Container Sanity Checking

This section will do some basic checking of your container parameters in STEP 6 against the hosts.

Sanity Check	Check Status	Description
	GOOD	yarn.scheduler.maximum-allocation-vcores must be greater than or equal to yarn.scheduler.minimum-allocation-vcores
	GOOD	yarn.scheduler.maximum-allocation-mb must be greater than or equal to yarn.scheduler.minimum-allocation-mb
	GOOD	yarn.scheduler.minimum-allocation-vcores must be less than or equal to the yarn.nodemanager.resource.cpu-vcores

STEP 7: MapReduce Configuration

Property	Property Type	Component	Value	Description
yarn.app.mapreduce.am.resource.cpu-vcores	Config	Application Master	1	AM container vcore reservation
yarn.app.mapreduce.am.resource.mb	Config	Application Master	1024	AM container memory reservation
mapreduce.map.cpu.vcores	Config	Map Task	1	Map task vcore reservation
mapreduce.map.memory.mb	Config	Map Task	1024	Map task memory reservation
mapreduce.reduce.cpu.vcores	Config	Reduce Task	1	Reduce task vcore reservation
mapreduce.reduce.memory.mb	Config	Reduce Task	1024	Reduce task memory reservation
mapreduce.task.io.sort.mb	Config	Spill/Sort (Map Task)	256	Spill/Sort memory reservation

Skalierbarkeit

- Empfehlung: 2-3 tasks per CPU Core!
- Dank AKKA können Spark-Tasks sehr kurz sein (>= 200ms)
- Nahezu lineare Skalierbarkeit

- http://spark.apache.org/docs/latest/tuning.html
- https://blog.cloudera.com/blog/2010/12/a-profile-of-hadoop-mapreduce-computing-efficiency-continued/

Inhalte

- Einführung
  - Anwendungsbereiche von Spark
  - Was ist Map-Reduce
  - Was sind RDDs
- Apache Spark im Detail
  - Spark Komponenten
  - Architekturübersicht
- Ausführen auf einem Cluster
  - Mesos
  - Standalone
  - Hadoop
- Spark vs Hadoop
  - Popularität Spark (+ Mesos) vs Hadoop
  - API Komplexität
  - Performance
  - Skalierbarkeit
- **Beispiele**

Beispiel SQL