Spezielle Kapitel Sozialer Web – **Technologien** – Spark Einführung

Seminaristischer Unterricht

Prof. Dr.-Ing. Hendrik Gärtner

Gliederung

- Motivation für Spark
- Einführung in die Programmierung
  - Spark Context
  - CRUD-Operationen
  - Indexierung
  - MapReduce

Das „Big Data-Problem“

- Datenmenge steigt wesentlich schneller als die Ge schwindigkeit der Prozessoren
- Immer mehr Datenquellen: Web, Mobile Geräte, Wissenschaft (Genome-Projekte, LHC), Industrie 4.0 ...
- Speicher wird immer billiger
- Speichergrößen verdoppeln sich alle 18 Monate
- Lesegeschwindigkeit der Speichergeräte verursachen häufig ein Ausbremsen der CPU

Big Data Examples

- Facebook’s täglich Logs: 60 TB
- 1,000 Genomes Projekte: 200 TB
- Google Web Index: 10+ PB
- Kosten für 1 TB of disk: ~$35

- Lesezeit von 1 TB von der Festplatte:
  3 Stunden ! (100 MB/s)

Das „Big Data-Problem“

Ein einzelner handelsüblicher Rechner kann die Datenmengen nicht in angemessener verarbeiten:

- Strategie Skalierung[1]:
  - Vertikale Skalierung (Scale up): Hinzufügen von Ressourcen zu einem Rechner (z.B. Speicher, CPUs,...)
  - Horizontale Skalierung (Scale out): Hinzufügen von Rechnern zu einem Cluster

[1]Unter Skalierbarkeit versteht man unter anderem in der Elektronischen Datenverarbeitung die Fähigkeit eines Systems aus Hard- und Software, die Leistung durch das Hinzufügen von Ressourcen – z. B. weiterer Hardware – in einem definierten Bereich proportional (bzw. linear) zu steigern.

Hardware für Big Data

- Zusammenschalten vieler „einfacher“ Hardwarekomponenten zu einem Cluster
- Keine High Endkomponenten sondern Rechenleistung wird durch die Anzahl der Computer gesteuert
- Vorteile:
  - Skalierung immer möglich
  - Preis der Hardware gering
- Nachteile:
  - Komplexität der Software extrem hoch

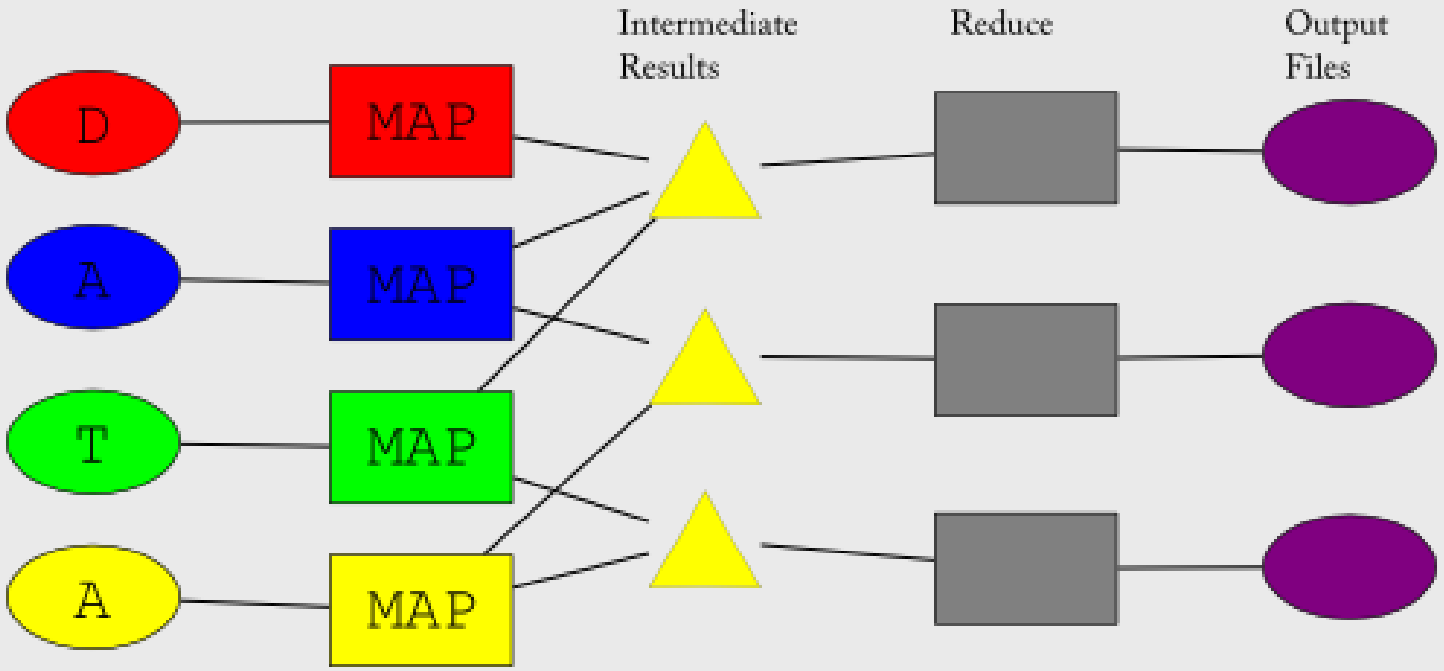
Probleme mit billiger Hardware

- Fehler in der Hardware (Zahlen von Google):
  - 1-5% Festplatten/Jahr
  - 0.2% DIMMs/Jahr
- Netzwerk Geschwindigkeit versus Shared Memory
  - Netzwerk hat viel mehr Latenzzeiten
  - Netzwerk ist wesentlich langsamer als der Speicher
- Ungleiche Performanz der einzelnen Maschinen

Problemstellungen des Cluster Computings

- Wie wird die Arbeit zwischen den Maschinen verteilt?
  - Data Locality: Welche Daten befinden sich auf welchen Rechnern?
  - Wie „weit“ sind Daten im Cluster entfernt? Wie ist das Netzwerk aufgebaut?
  - Wie „teuer“ ist es, Daten zu verschieben?
- Wie soll mit Fehlern umgegangen werden?
  - Wenn 1 Server alle 3 Jahre ausfällt, dann hat ein Cluster mit 10.000 Knoten jeden Tag durchschnittlich 10 Ausfälle
  - Wie soll mit sogenannten „Stragglers“ (kein Ausfall, nur sehr langsam) umgegangen werden? Wie werden Sie erkannt und wie soll darauf reagiert werden?

Verteiltes MapReduce



Ansatz Hadoop:

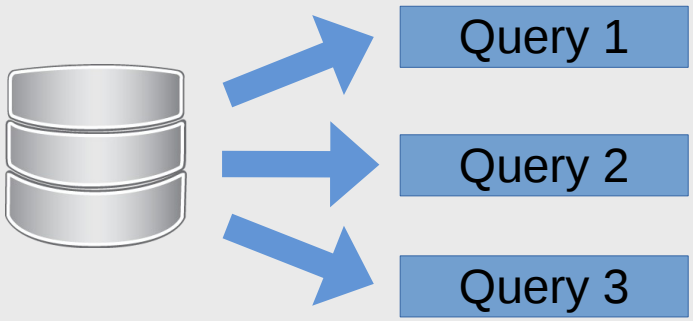
- In jedem Schritt werden Ergebnisse auf der Festplatte abgelegt
- Im nächsten Schritt werden die Daten wieder von der Festplatte gelesen
- Hadoop ist auf die MapReduce-Anwendung festgelegt

Map Reduce: Iterative Jobs

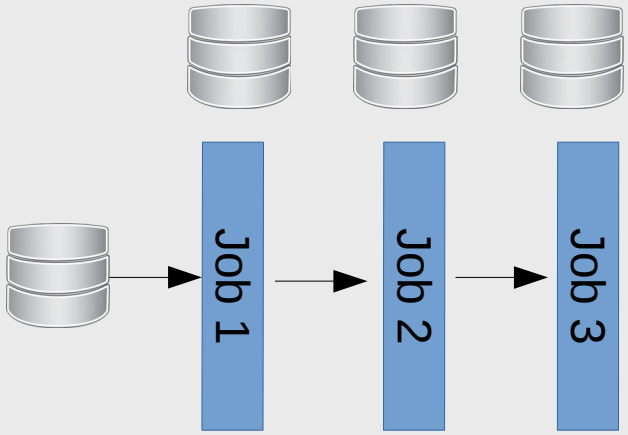
- Iterative Jobs (z.B. Page Rank) benötigen extrem viel Disk IO
- Disk IO ist kostspielig

Motivation für Apache Spark

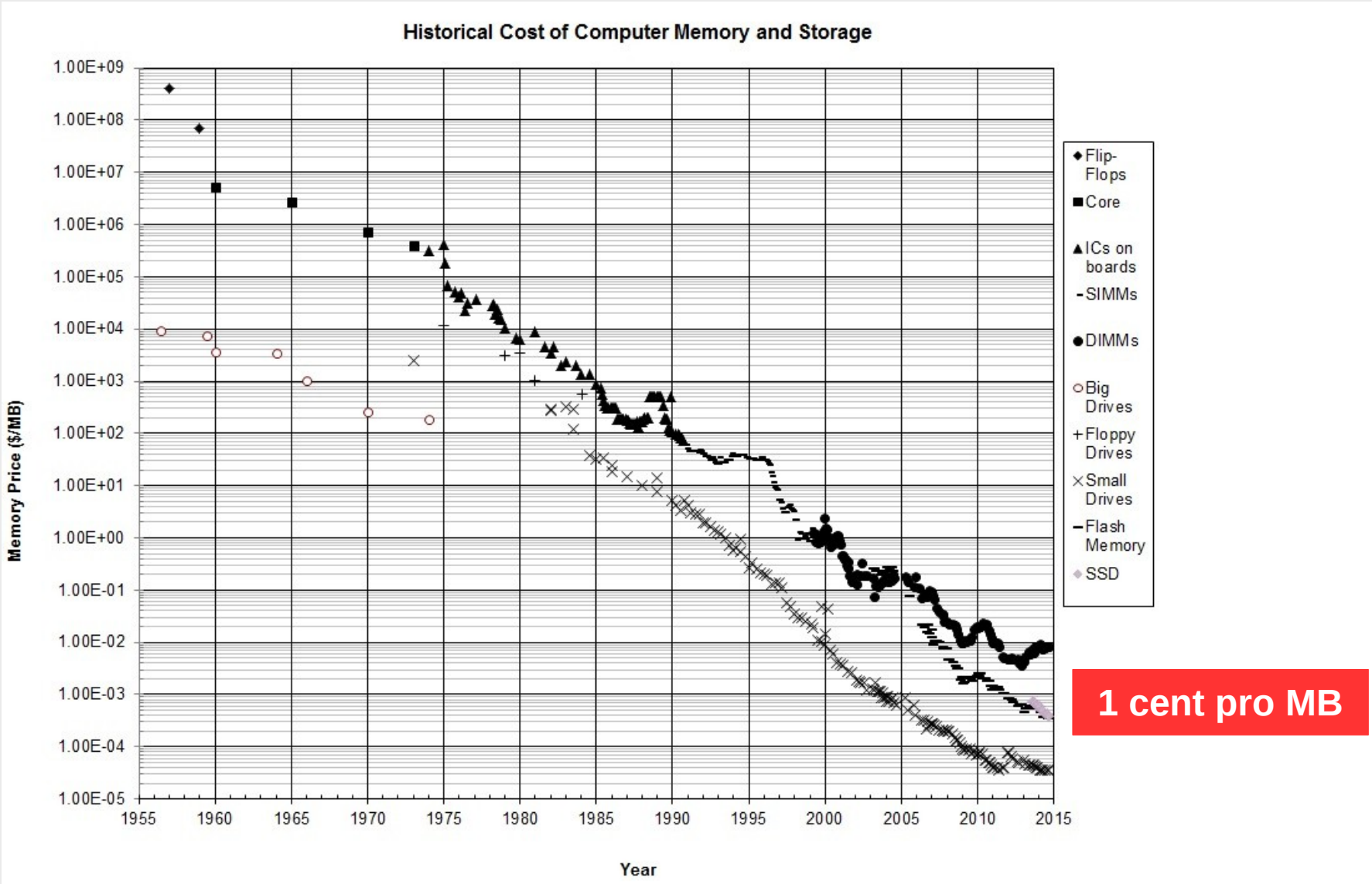
- Einsatz von Map Reduce für komplexe Anfragen, Iterative Jobs oder Online-Processing erfordern viel Disk-IO
- Disk IO ist langsam, so dass Map Reduce für zeitkritische Systeme gar nicht eingesetzt werden kann



Interactive Mining



Stream Processing

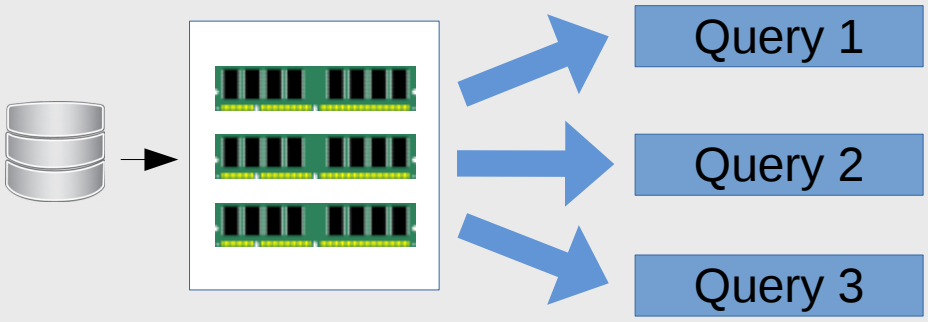


1 cent pro MB

http://www.jcmit.com/mem2014.htm

In Memory Data Sharing

- Ansatz von Spark: Halte möglichst viele Daten im Hauptspeicher vor, so dass kein Disk-IO stattfindet
- Teile die Daten auf und halte sie als „Distributed Memory“ auf mehreren Rechnern vor
- Verarbeite die Daten möglichst nur im Hauptspeicher
- Spark ist dadurch 10-100x schneller als Hadoop



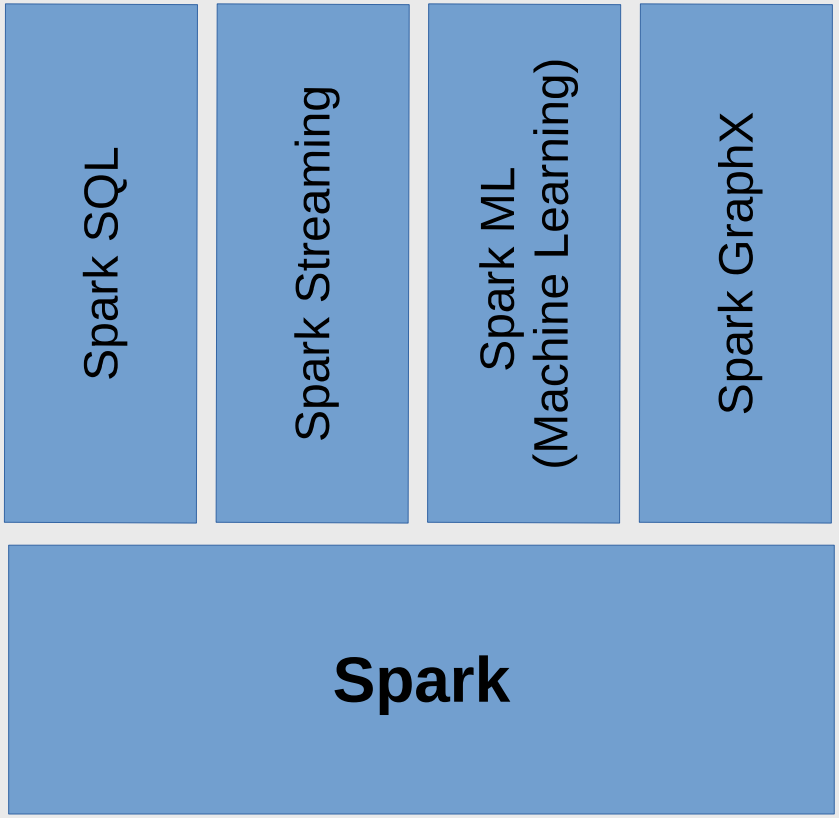
Resilient Distributed Dataset

- RDDs abstrahieren von den großen, verteilt gespeicherten Daten
- Programmer werden auf Basis von Operationen auf den Verteilten Datenmengen geschrieben (map, filter,...)
- Partitionierte Collections werden über einen Cluster verteilt und entweder im Speicher oder auf der Disk gehalten
- RDDs werden erzeugt und manipuliert von „Parallelen Transformationen“ (map, filter, join) und sogenannten Actions (count, collect, save)
- RDDs werden automatisch wiederhergestellt im Fehlerfalle

Das Spark Framework

- Stellt eine Programmierabstraktion und eine parallele Laufzeitumgebung, die die Komplexität der Fehlertoleranz und Langsamen Maschinen verbirgt
- „Hier ist eine Operation, führe sie auf der gesamten Datenmenge aus“
  - Es interessiert mich nicht, wo die Operationen ausgeführt wird.
  - Wenn erforderlich, führe sie mehrfach auf verschiedenen Knoten aus.

Das Spark Framework



Unterschied Hadoop MR und Spark

	Hadoop MR	Spark
Storage	Disk only	In Memory, Disk
Operations	Map, Reduce	Map, Reduce, Filter, Join,...
Execution Models	Batch	Batch, Interactive, Streaming
Envrironments	Java	Scala, Java, Python, R

Spark führt derzeit, beim Sortierbenchmark – schnellstes Framework: https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Weitere Vorteile von Spark gegenüber Hadoop

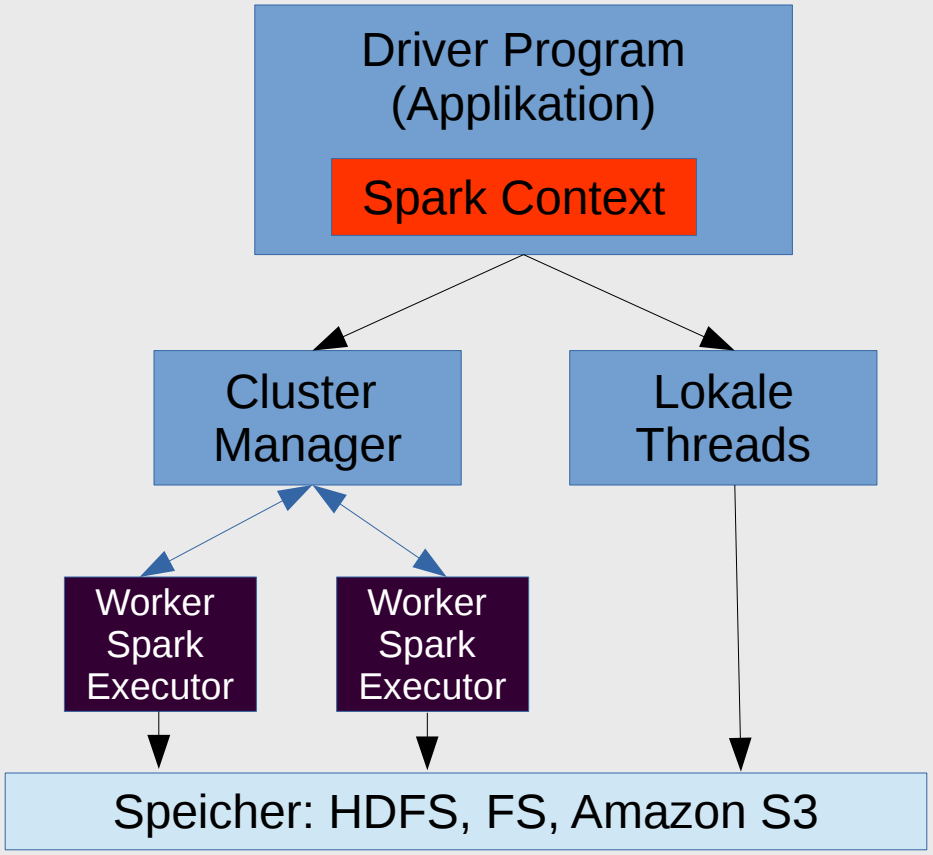
- Spark implementiert generalisierte Patterns
- Einheitliche Engine, die für viele Anwendungsfälle genutzt werden kann
- Lazy evaluation des Abstammungsgraphen eines RDDs
- Reduzierte Wartezyklen, besseres Pipeling
- Weniger Overhead für das Starten der Jobs
- Weniger kostspielige Shuffle Operationen

Programmierung von Spark

Spark Architektur

Eine Spark-Applikation besteht aus zwei Programmen:

- Ein sogenannter Treiber (Driver Program) sowie die
- Worker, die die eigentliche Arbeit verrichten



Ein Worker läuft entweder:

- innerhalb eines Clusters oder
- in Localen Threads

Die RDDs sind verteilt

Spark Context

- Ein Spark Programm erzeugt als erstes einen sogenannten SparkContext
- Der SparkContext weiß, wie auf den Cluster zugegriffen werden kann
- Die Scala-Shell erzeugt automatisch einen SparkContext, der über die Variable sc zugegriffen werden kann
- Scala Programme müssen diesen mittels new und einem entsprechenden Konstruktor erzeugen
- Über den SparkContext können dann RDDs erzeugt werden

Spark Essentials: Der Master-Parameter

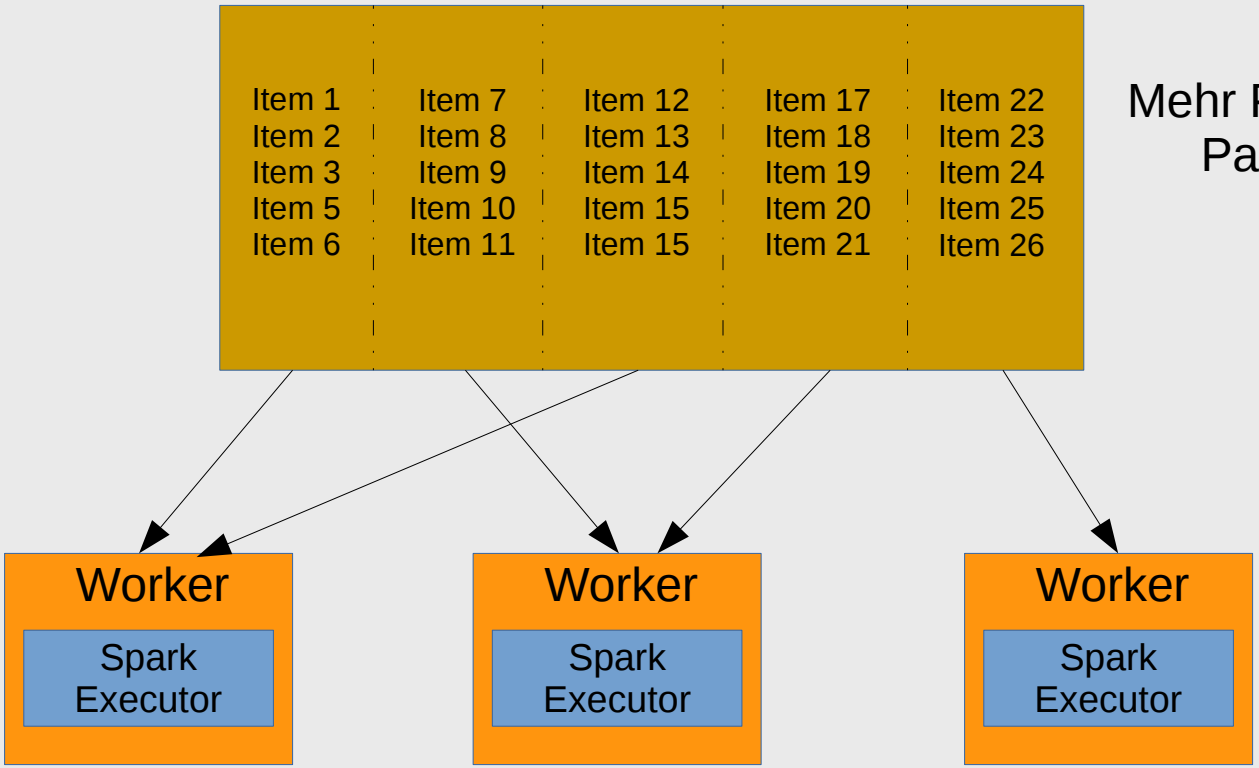
- Über den Master-Parameter wird bestimmt, wie auf den Cluster zugegriffen werden kann:
- local: Spark wird lokal mit einem Worker verwendet (Ohne Parallelisierung)
- local[X]: Spark wird lokal mit X-Worker Prozessen (idealerweise entspricht X der Anzahl der Cores) gestartet
- spark:HOST:PORT: Verbindung zu einem Standalone Spark-Cluster
- mesos:HOST:PORT: Verbindung zu einem Mesos-Cluster

Resilient Distributed Datasets

- Die Primäre Abstraktion in Spark
- Sind Immutable wenn sie erzeugt wurden
- Speichern Herkunftsinformationen, um sie effizient neu berechnen zu können bei Verlust
- Erlauben Operationen auf Collections, die parallel ausgeführt werden
- Können durch folgende Operationen erzeugt werden:
  - Parallelisierung existierender Collections,
  - Transformation eines bestehenden RDDs und
  - aus Files im HDFS oder vielen anderen Storage Sytemen.

Resilient Distributed Dataset

Programmierer spezifiziert die Nummer der Partitionen des RDDs



Mehr Partitionen mehr Parallelisierung

Operationen auf RDDs

- Spark bietet zwei Typen von Operationen: Transformationen und Actions
- Transformation sind „lazy“ (werden erst berechnet, wenn erforderlich)
- Transformationen auf dem RDD werden ausgeführt, wenn eine Action aufgerufen wird
- Persist (Cache) RDDs werden im Memory oder auf der Disk gecacht

Arbeiten mit RDDs

1) Erzeugen eines RDDs über eine Datenquelle
2) Verändern des RDDs mittels der Transformationen
3) Sammeln des Ergebnisses durch das Durchführen einer Action

Beispiel

1) val l= 1 to 1000000000 (Anlegen einer Range – wird nicht aufgelöst)
2) l.parallelize(l,8) (Erzeugen eines RDDs mit 8 Partitionen)
3) val res= rdd.filter(X=>X %3 ==0).map (X=>Math.sqrt(X)) (Transformationen auf dem RDD)
4) res.take(10) (Action auf dem RDD)
5) res.collect (Action auf dem RDD)

Beispiel Twitter-Analyse

```
{
    "text": "RT @PostGradProblem: In preparation for the NFL lockout, I will be spending twice as
    much        time analyzing my fantasy baseball team during ...",
    “truncated": true,
    "in_reply_to_user_id": null,
    “in_reply_to_status_id": null,
    "favorited": false,
...
    "entities": {
        "user_mentions": [
            {
                "indices": [3,19],
                "screen_name": "PostGradProblem",
                “id_str": "271572434",
                "name": "PostGradProblems",
                "id": 271572434
            }
        ],
        "urls": [ ],
        "hashtags": [ ]
...
```

Extraktion von Tweets

```
import scala.util.parsing.json.{JSON, JSONObject}

def extractTweet(doc:String):List[String]={

    val tweet= JSON.parseFull(doc)

    tweet match{
    case Some(m)=> val map= m.asInstanceOf[Map[String,Any]];
    (map.get("text"), map.get("lang")) match {


    case (Some(txt:String), Some("de")) => List(txt)
    case _ => List()

    }

    case None=> List()

    }

 }
```

Achtung: Keine JSON-Unterstützung in Scala 2.11

Einsatz

- Einsatz auf einem Rechner:
  spark-shell --master local[4]
  val rdd=sc.textFile(“file://home/hgaertner/spark/twitter/tweetsbig.txt“)
  val res= rdd.flatMap(X=>extractTweet(X))
  res.take(10)
  val fullres= res.collect

Dauer: ca. 30min

- Einsatz im Cluster:
  spark-shell --master spark://hadoop03.f4.htw-berlin.de
  val rdd=sc.textFile(“tweetsbig.txt“)
  val res= rdd.flatMap(X=>extractTweet(X))
  res.take(10)
  val fullres= res.collect

Dauer: ca. 9 min

Im Cluster wird standardmäßig auf das HDFS zugegriffen.

Typische Transformationen

- **map(func):**
  Erzeugt ein neues RDD unter Anwendung der übergebenen Funktion auf jedes Element
- **filter(func) :**
  Erzeugt ein neues RDD, in dem alle Elemente überprüft werden, ob sie das durch die übergebene Funktion repräsentiertes Prädikat erfüllen oder nicht. Wenn ja, wird der Datensatz übernommen, wenn nein nicht.
- **distinct([numTasks])):**
  Erzeugt ein neues RDD durch das Löschen aller Duplikate
- **flatMap(func):**
  Funktion wie map, nur zusätzliches flachziehen auf einer Ebene.

Typische Actions

- **reduce(func):**
  Aggregiert eine Datemenge unter Verwendung der angegebenen Funktion. Die Funktion muss zwei Argumente vom selben Typ bekommen sowie Asoziativ und Kommutatik sein, damit sie parallel ausgeführt werden kann.
- **take(n):**
  Gibt ein Array mit den ersten n Elementen zurück.
- **collect()**
  Gibt einen Array mit allen Elementen zurück.
  Achtung: Das resultierende Array muss in den Hauptspeicher des Driver-Programms passen.
- **takeOrdered(n)(implicit ordering):**
  Gibt einen Array bestehend aus n-Elementen zurück, wobei diese in sortierter Reihenfolge ausgewählt werden

Es gibt viele weitere Funktionen (z.B. groupBy, reduceByKey, aggregate,...– siehe Dokumentation).

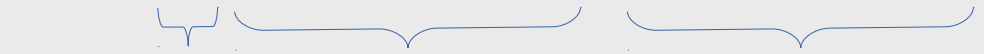
Beispiel

- Erzeugen von Zufallszahlen:
  val data= for (i <- 1 to 1000) yield rand.nextInt(100)
- Umwandeln in ein RDD:
  val rdd= sc.parallelize(data,4)
- Zählen der Vorkommen:
  val res=rdd.groupBy(X=>X).map(X=>(X._1,X._2.size)) oder
  val res= rdd.map(X=>(X,1)).reduceByKey((X,Y)=>X+Y)
- Ermitteln der 10 Zahlen, die am häufigsten vorkommen:
  res.takeOrdered(10)(Ordering[Int].on(X=>(-1)*X._2))

Achtung: Aggregation/Reduzierung

- Eigentlich zwei Funktionen erforderlich:
  - Reduzierung der Partition
  - Zusammenfassung der Partitionen
- Reduce nimmt dieselben Funktionen
- Aggregate By Key verwendet unterschiedliche Funktionen

```
val rddmap= rdd.map(X=>(X,1))
rddmap.aggregateByKey(0)((X,Y)=>X+1, (A,B)=>A+B)
```



Basiswert | Zusammenfassung Partition | Zusammenfassung Partitionen

Spark SQL - Row

- Repräsentiert eine Output-Zeile eines Relationalen Operators
- Rows bieten die Möglichkeit Werte unterschiedlichen Typs in einem Datensatz zusammenzufassen
- Rows haben Methoden zum Zugriff der einzelnen Elemente, dabei kann automatisch eine Konvertierung zum entsprechen Datentyp erfolgen: z.B. getInt(X) oder getString(X) holt den X.ten Wert und konvertiert diesen
- Die Methode get(X) holt den X.ten Wert – eine Konvertierung zu einem best. Typen erfolgt über die Methode asInstanceOf[*Typ*]

Vielen Dank für

Ihre Aufmerksamkeit

Prof. Dr.-Ing. Hendrik Gärtner