The Stratosphere Big Data Analytics Platform

Amir H. Payberah
Swedish Institute of Computer Science

amir@sics.se

June 4, 2014

Big Data

small data

big data

- Big Data refers to datasets and flows large enough that has outpaced our capability to store, process, analyze, and understand.

Where Does Big Data Come From?

Big Data Market Driving Factors

The number of web pages indexed by Google, which were around one million in 1998, have exceeded one trillion in 2008, and its expansion is accelerated by appearance of the social networks.∗

∗ "Mining big data: current status, and forecast to the future" [Wei Fan et al., 2013]

Big Data Market Driving Factors

The amount of mobile data traffic is expected to grow to 10.8 Exabyte per month by 2016.∗

∗ "Worldwide Big Data Technology and Services 2012-2015 Forecast" [Dan Vesset et al., 2013]

Big Data Market Driving Factors

More than 65 billion devices were connected to the Internet by 2010, and this number will go up to 230 billion by 2020.∗

∗ "The Internet of Things Is Coming" [John Mahoney et al., 2013]

Big Data Market Driving Factors

Many companies are moving towards using Cloud services to access Big Data analytical tools.

Big Data Market Driving Factors

Open source communities

APACHE HBASE

mahout

GraphX

hadoop

StratoSphere
Above the Clouds

Storm

GraphLab

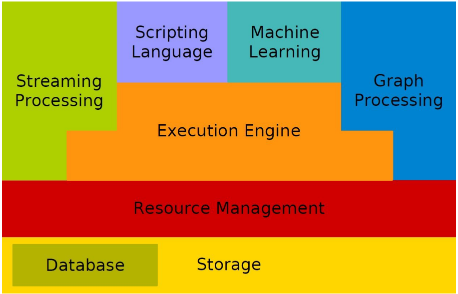
S4 distributed stream computing platform

HIVE

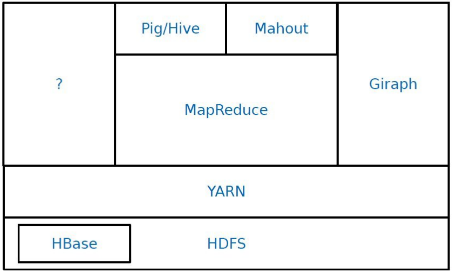
Spark

cassandra

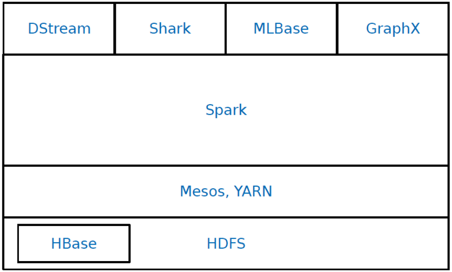
Big Data Analytics Stack



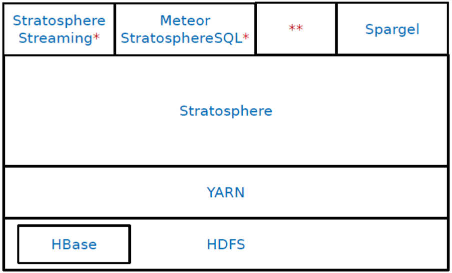
Hadoop Big Data Analytics Stack



Spark Big Data Analytics Stack



Stratosphere Big Data Analytics Stack



* Under development

** Stratosphere community is thinking to integrate with Mahout

Big Data looks tiny from Stratosphere

What is Stratosphere?

- An efficient distributed general-purpose data analysis platform.
- Built on top of HDFS and YARN.
- Focusing on ease of programming.

Project Status

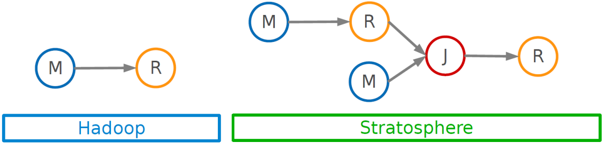
- Research project started in 2009 by TU Berlin, HU Berlin, and HPI
- An Apache Incubator project
- 25 contributors
- v0.5 is released

Motivation

- MapReduce programming model has not been designed for complex operations, e.g., data mining.
- Very expensive, i.e., always goes to disk and HDFS.

Solution

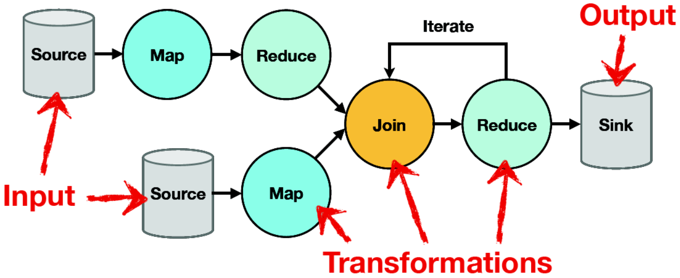
- Extends MapReduce with more operators.
- Support for advanced data flow graphs.
- In-memory and out-of-core processing.



Stratosphere Programming Model

Stratosphere Programming Model

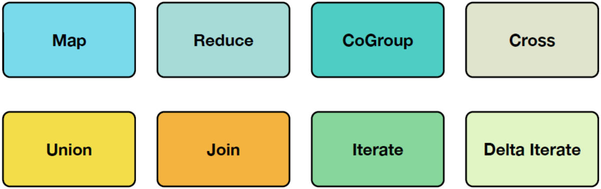
- A program is expressed as an arbitrary data flow consisting of transformations, sources and sinks.



Transformations

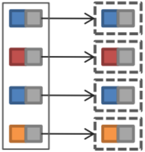
- Higher-order functions that execute user-defined functions in parallel on the input data.

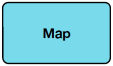
- Higher-order functions that execute user-defined functions in parallel on the input data.



Transformations: Map

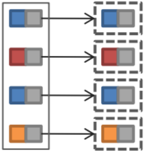
- All pairs are independently processed.



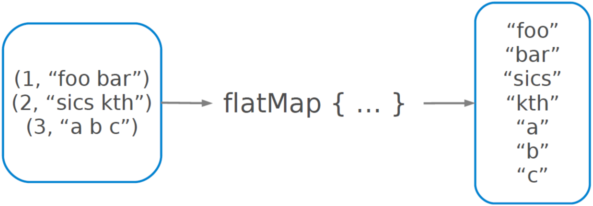


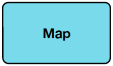
Transformations: Map

- All pairs are independently processed.



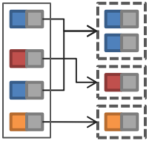
```scala
val input: DataSet[(Int, String)] = ...

val mapped = input.flatMap { case (value, words) => words.split(" ") }
```

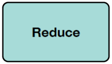




Transformations: Reduce

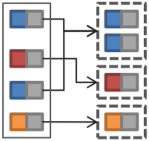
- Pairs with identical key are grouped.
- Groups are independently processed.



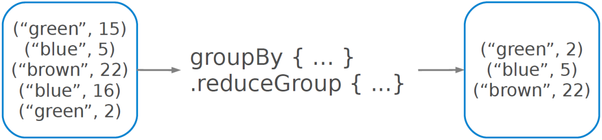


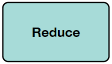
Transformations: Reduce

- Pairs with identical key are grouped.
- Groups are independently processed.



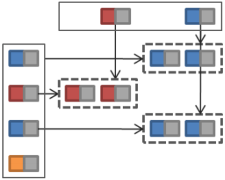
```
val input: DataSet[(String, Int)] = ...

val reduced = input.groupBy(_._1)
                   .reduceGroup(words => words.minBy(_._2))
```



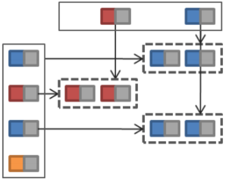


Transformations: Join

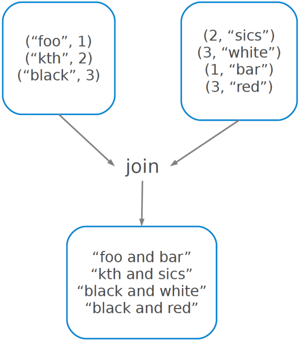
- Performs an equi-join on the key.
- Join candidates are independently processed.



Transformations: Join



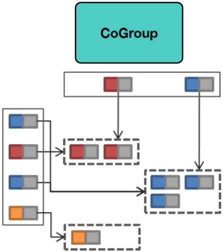
- Performs an equi-join on the key.
- Join candidates are independently processed.




```
val counts: DataSet[(String, Int)] = ...
val names: DataSet[(Int, String)] = ...

val join = counts.join(names)
  .where(_._2)
  .isEqualTo(_._1)
  .map { case (l, r) => l._1 + "and" + r._2 }
```

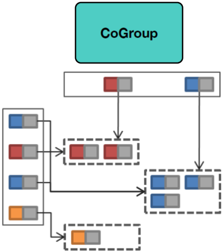
Join

Transformations: and More ...

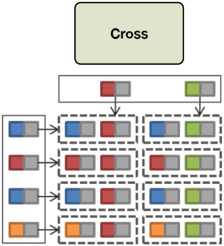


Groups each input on key

Transformations: and More ...

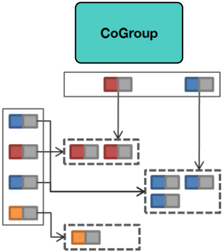


Groups each input on key

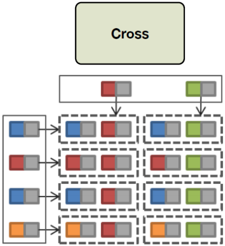


Builds a Cartesian Product

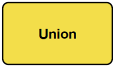
Transformations: and More ...



Groups each input on key



Builds a Cartesian Product



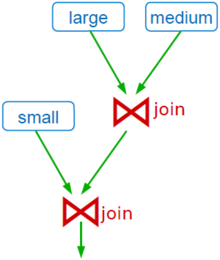
Merges two or more input data sets, keeping duplicates

Example: Word Count

```
val input = TextFile(textInput)

val words = input.flatMap { line => line.split(" ")
                            map(word => (word, 1)) }

val counts = words.groupBy(_._1)
                  .reduce((w1, w2) => (w1._1, w1._2 + w2._2))

val output = counts.write(wordsOutput, CsvOutputFormat())
```

Join Optimization

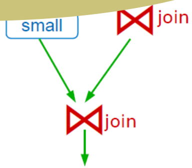
```
val large = env.readCsv(...)
val medium = env.readCsv(...)
val small = env.readCsv(...)

joined1 = large.join(medium)
               .where(_._3)
               .isEqualTo(_._1)
               .map { (left, right) => ... }

joined2 = small.join(joined1)
               .where(_._1)
               .isEqualTo(_._2)
               .map { (left, right) => ...}

result = joined2.groupBy(_._3)
                .reduceGroup(_.maxBy(_._2))
```



Join Optimization

```
val large = env.readCsv(...)
val medium = env.readCsv(...)
val small = env.readCsv(...)

joined1 = large.join(medium)
               .where(_._3)
               .isEqualTo(_._1)
               .map { (left, right) => ... }

joined2 = small.join(joined1)
               .where(_._1)
               .isEqualTo(_._2)
               .map { (left, right) => ...}

result = joined2.groupBy(_._3)
                .reduceGroup(_.maxBy(_._2))
```

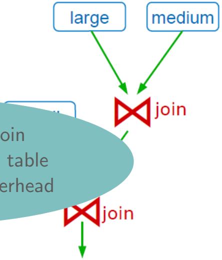
(1) Partitioned hash-join
- Hash/Sort tables into N buckets on join key
- Send buckets to parallel machines
- Join every pair of buckets



Join Optimization

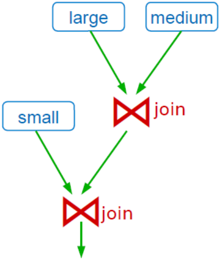
```
val large = env.readCsv(...)
val medium = env.readCsv(...)
val small = env.readCsv(...)

joined1 = large.join(medium)
               .where(_._3)
               .isEqualTo(_._1)
               .map { (left, right) =>

joined2 = small.join(joined1)
               .where(_._1)
               .isEqualTo(_._2)
               .map { (left, right) => ...}

result = joined2.groupBy(_._3)
                .reduceGroup(_.maxBy(_._2))
```

(2) Broadcast hash-join
- Broadcast the smaller table
- Avoids the network overhead



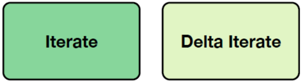
Join Optimization

```
val large = env.readCsv(...)
val medium = env.readCsv(...)
val small = env.readCsv(...)

joined1 = large.join(medium)
               .where(_._3)
               .isEqualTo(_._1)
               .map { (left, right) => ... }

joined2 = small.join(joined1)
               .where(_._1)
               .isEqualTo(_._2)
               .map { (left, right) => ...}

result = joined2.groupBy(_._3)
                .reduceGroup(_.maxBy(_._2))
```



(3) Grouping/Aggregation reuses the partitioning from step (1) - no shuffle

What about Iteration?

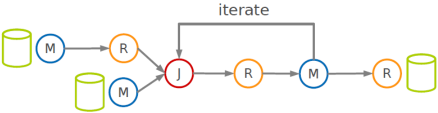


Iterative Algorithms

- Algorithms that need iterations:
  - Clustering, e.g., K-Means
  - Gradient descent, e.g., Logistic Regression
  - Graph algorithms, e.g., PageRank
  - ...

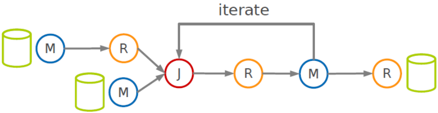
Iteration

- Loop over the working data multiple times.

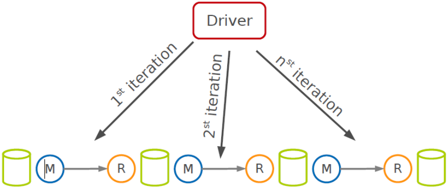


Iteration

- Loop over the working data multiple times.



- Iterations with hadoop
  - Slow: using HDFS
  - Everything has to be read over and over again



Types of Iterations

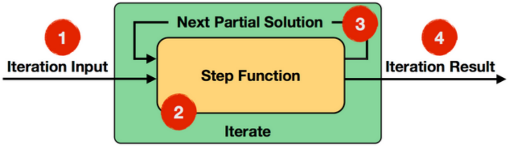
- Two types of iteration at stratosphere:
  - Bulk iteration
  - Delta iteration

- Both operators repeatedly invoke the step function on the current iteration state until a certain termination condition is reached.

Bulk Iteration

- In each iteration, the step function consumes the entire input, and computes the next version of the partial solution.

- A new version of the entire model in each iteration.



Bulk Iteration - Example

```
// 1st           2nd                    10th
map(1) -> 2      map(2) -> 3      ...   map(10) -> 11
map(2) -> 3      map(3) -> 4      ...   map(11) -> 12
map(3) -> 4      map(4) -> 5      ...   map(12) -> 13
map(4) -> 5      map(5) -> 6      ...   map(13) -> 14
map(5) -> 6      map(6) -> 7      ...   map(14) -> 15
```

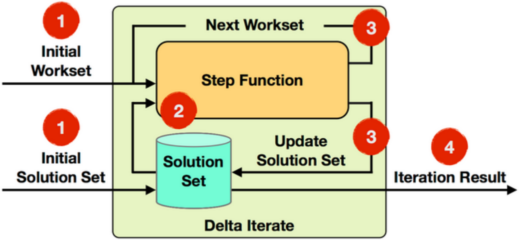
Bulk Iteration - Example

```
// 1st          2nd                          10th
map(1) -> 2     map(2) -> 3      ...         map(10) -> 11
map(2) -> 3     map(3) -> 4      ...         map(11) -> 12
map(3) -> 4     map(4) -> 5      ...         map(12) -> 13
map(4) -> 5     map(5) -> 6      ...         map(13) -> 14
map(5) -> 6     map(6) -> 7      ...         map(14) -> 15
```

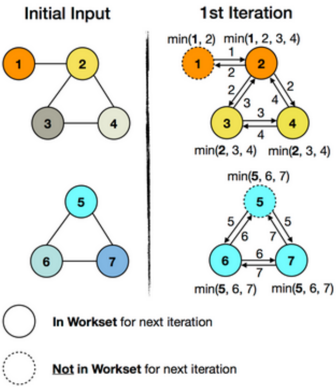
```
val input: DataSet[Int] = ...

def step(partial: DataSet[Int]) = partial.map(a => a + 1)

val numIter = 10;
val iter = input.iterate(numIter, step)
```

Delta Iteration

- Only parts of the model change in each iteration.

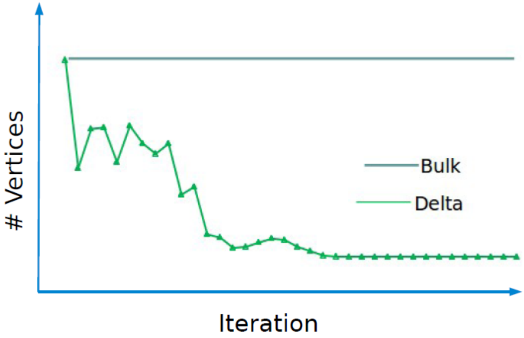


Delta Iteration - Example



Delta Iteration vs. Bulk Iteration

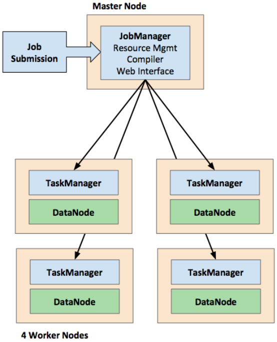
- Computations performed in each iteration for connected communities of a social graph.



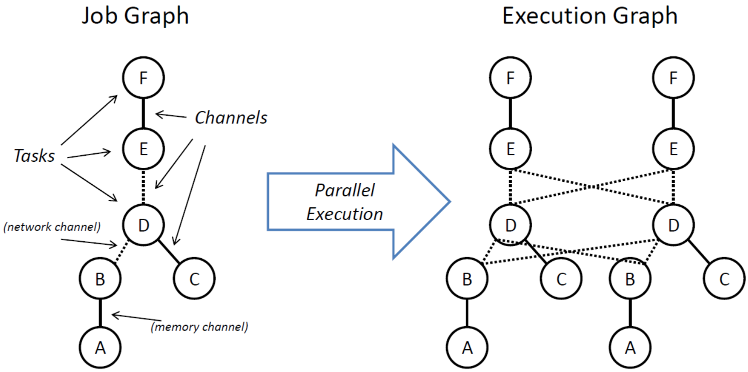
Stratosphere
Execution Engine

Stratosphere Architecture

- Master-worker model
- Job Manager: handles job submission and scheduling.
- Task Manager: executes tasks received from JM.
- All operators start in-memory and gradually go out-of-core.



Job Graph and Execution Graph

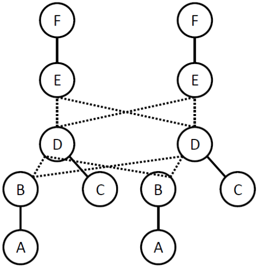


- Jobs are expressed as data flows.
- Job graphs are transformed into the execution graph.
- Execution graphs consist information to schedule and execute a job.

Channels

- Channels transfer serialized records in buffers.
- Pipelined
  - Online transfer to receiver.
- Materialized
  - Sender writes result to disk, and afterwards it transferred to receiver.
  - Used in recovery.

Execution Graph



Fault Tolerance

- Task failure compensated by backup task deployment.
- Track the execution graph back to the latest available result, and recompute the failed tasks.

	Hadoop	Stratosphere	Spark
Paradigm	MapReduce	Iterative Data Model	Distributed Collections (RDD)
Data Model	Key/Value Pairs	Tuples	Key/Value Pairs
Runtime	Batch Processing	Streaming in-memory and out-of-core	Batch Processing in-memory and out-of-core
Compilation Optimization	None	Holistic Planning for Data Exchange, Sort/Hash, Caching, ...	?

http://stratosphere.eu