

Stratosphere for Hadoop Users

Potsdam, January 03, 2012

Arvid Heise

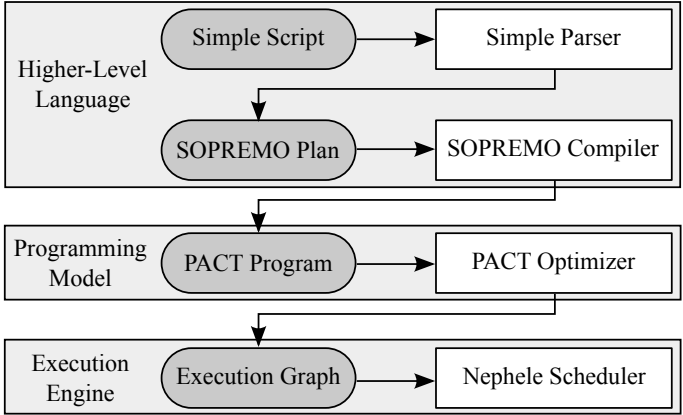
Outline

2

- 1 Overview over Stratosphere
- 2 Dataflow Orientation
- 3 Tuple-based Data Model
- 4 Other Differences
- 5 Seminar Organization

Stratosphere Stack

3



- Parallel programming model
- Implementation and generalization of Map/Reduce
- Similar interface as Hadoop
- Defines the parallelization semantics of tasks
- Pact plan is dataflow-oriented
- Pact optimizes plans and compiles them to execution graphs for Nephelè

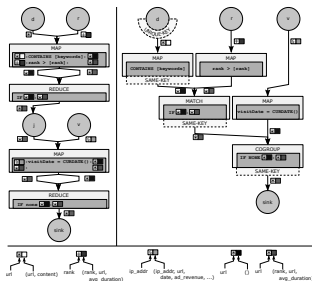
- Alexandrov et al. 2010. MapReduce and Pact - Comparing Data Parallel Programming Models.

Hadoop and Stratosphere Job

5

```

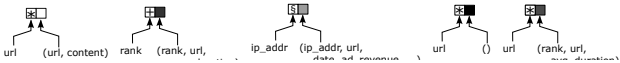
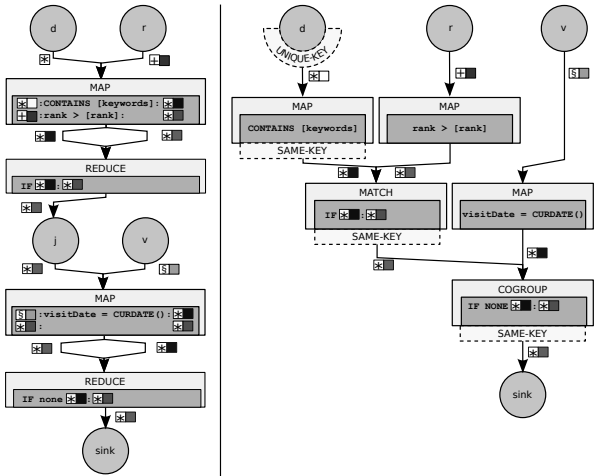
SELECT * FROM Documents d JOIN Rankings r ON r.url = d.url
WHERE CONTAINS(d.text, [keywords]) AND r.rank > [rank]
AND NOT EXISTS (SELECT * FROM Visits v WHERE v.url = d.url
AND v.visitDate = CURDATE());
```



- Battré et al. 2010. Nephelè/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing

Hadoop and Stratosphere Job

5



Nephele

6

- Executes an Execution Graph
- Decides for each task how many instances are appropriate
- Assigns task instances to computation units
- Manages fault tolerance and adapts to changes

- Daniel Warneke and Odej Kao. 2009. Nephele: Efficient Parallel Data Processing in the Cloud

Sopremo and Simple

7

- High level language layer
- Simple = query language
- Sopremo = semi-structured data model (JSON) and operators
- Extensible operators for several use cases
- Text Mining, Data Cleansing, Data Mining

Outline

8

- 1 Overview over Stratosphere
- 2 Dataflow Orientation**
- 3 Tuple-based Data Model
- 4 Other Differences
- 5 Seminar Organization

Data Analysis Program

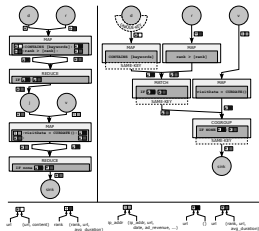
9

Hadoop

- Driver program + multiple jobs
- Driver program manually connects input and outputs
- Fixed pipeline
- 1 Job = 1 Map + 1 Reduce

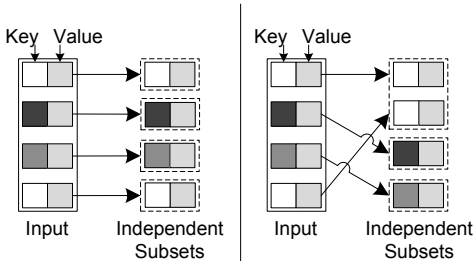
Stratosphere

- Directed acyclic graph of arbitrary Pacts
- Explicit data sources and sinks
- Pact also support two inputs (for join-like operations)



Map/Reduce in Stratosphere

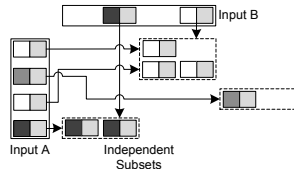
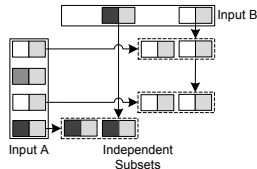
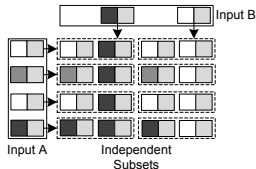
- Works same as Hadoop
- But the semantics are interpreted differently
- Each Pact defines data dependencies
- Map: each tuple can be treated separately
- Reduce: tuple with same key are grouped and guaranteed to be processed by same reducer



Two Input Pacts

11

- Currently three additional Pacts for two inputs
- Cross: make all possible pairs
- Match: find all matching pairs
- CoGroup: group all matching tuples
- All pairs/groups are treated independently



Comparison

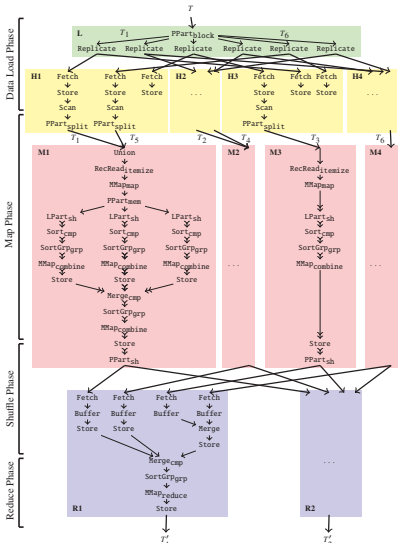
12

Hadoop

- Often workarounds necessary in a complex M/R program
- Error-prone, re-occurring manual data partitioning
- Pattern evolved for joins etc. (Data mining book)
- Fixed pipeline allows fine-grain tweaks (exploits)

Comparison

12



Comparison

12

Hadoop

- Often workarounds necessary in a complex M/R program
- Error-prone, re-occurring manual data partitioning
- Pattern evolved for joins etc. (Data mining book)
- Fixed pipeline allows fine-grain tweaks (exploits)

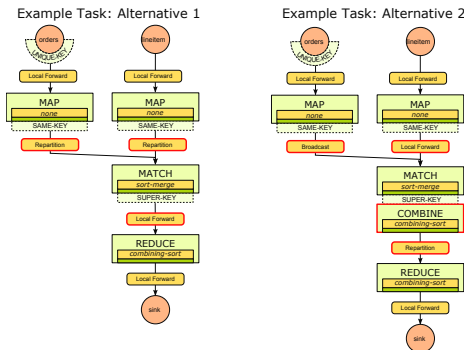
Stratosphere

- More expressiveness for complex data operations
- Maintains dataflow semantics to some degree
- Allows optimization and different shipping strategy
- Less hooks than Hadoop

Optimization

13

- Inspired by traditional query optimization
- Choose best join/shipping strategy for plan



- Reorder Facts

Outline

14

- 1 Overview over Stratosphere
- 2 Dataflow Orientation
- 3 Tuple-based Data Model**
- 4 Other Differences
- 5 Seminar Organization

Switch to Tuple-based Model

15

- Bleeding Edge! Check pact-examples subproject
- Typical Hadoop job:
 - Map: transforms/filters data, sets key
 - Reduce: combines data, unsets key
- Preceding maps limit reordering

Switch to Tuple-based Model

15

- Bleeding Edge! Check pact-examples subproject
- Typical Hadoop job:
 - Map: transforms/filters data, sets key
 - Reduce: combines data, unsets key
- Preceding maps limit reordering
- Stratosphere uses tuples instead of k/v-pairs
- User should set keys as soon as possible for ALL Pacts
- Each Pact is annotated to define the "key"

Reduce Example

16

```

1  public static class CountWords extends ReduceStub {
2      public void reduce(Iterator< PactRecord> records,
3          Collector out){
4          PactRecord element = null;
5          int sum = 0;
6          while (records.hasNext()) {
7              element = records.next();
8              PactInteger count =
9                  element.getField(1, PactInteger.class);
10             sum += count.getValue();
11         }
12         element.setField(1, new PactInteger(sum));
13         out.collect(element);
14     }
15 }
16 ...
17 ReduceContract reducer = new ReduceContract(
18     CountWords.class, // <-- UDF
19     PactString.class, // <-- key class
20     0, // <-- key index
21     mapper); // <-- input

```

PactRecords Semantics

17

- List of fields = keys or values
- Fields that are used as keys need to implement Key
- Maintains serialized form as long as possible, lazy deserialization
 - Fields are only deserialized when read
 - Serializes only written fields
 - Needs type of field to deserialize
- Very efficient storage of null values
- ⇒ Use a separate field for each key in your code

Word Count Example

18

- Wrap Pact Stubs in Pact Contracts
- All pacts except source specify their input

```
1 String dataInput = ...;
2 FileDataSource source = new FileDataSource(
3     LineInFormat.class, dataInput, "Input_Lines");
4 MapContract mapper = new MapContract(
5     TokenizeLine.class, source, "Tokenize_Lines");
6 ReduceContract reducer = new ReduceContract(
7     CountWords.class, PactString.class, 0, mapper,
8     "Count_Words");
9 FileDataSink out = new FileDataSink(
10    WordCountOutFormat.class, output, reducer,
11    "Word_Counts");
```

Word Count Example

18

- Input format needs to be parsed in any case
- Here we could already split lines and omit map
- In this case, we emit a PactRecord with 1 field
- Reuse objects to minimize garbage collection

```
1 public static class LineInFormat extends
    DelimitedInputFormat {
2     private final PactString string = new PactString();
3
4     public boolean readRecord(PactRecord record, byte[] line,
        int numBytes) {
5         this.string.setValueAscii(line, 0, numBytes);
6         record.setField(0, this.string);
7         return true;
8     }
9 }
```

Word Count Example

18

■ Implicit semantic of fields

```

1  public static class TokenizeLine extends MapStub {
2      private final PactRecord outputRecord = new PactRecord();
3      private final PactString string = new PactString();
4      private final PactInteger integer = new PactInteger(1);
5
6      private final AsciiUtils.WhitespaceTokenizer tokenizer =
7          new AsciiUtils.WhitespaceTokenizer();
8
9      @Override
10     public void map(PactRecord record, Collector collector) {
11         // get the first field (as type PactString)
12         PactString str = record.getField(0, PactString.class);
13
14         // tokenize the line
15         this.tokenizer.setStringToTokenize(str);
16         while (tokenizer.next(this.string)) {
17             // we emit a (word, 1) pair
18             this.outputRecord.setField(0, this.string);
19             this.outputRecord.setField(1, this.integer);
20             collector.collect(this.outputRecord);
21         }
22     }
23 }

```


Word Count Example

18

```
1  @Combinable
2  public static class CountWords extends ReduceStub {
3      private final PactInteger theInteger = new PactInteger();
4
5      @Override
6      public void reduce(Iterator<PactRecord> records,
7          Collector out) throws Exception {
8          PactRecord element = null;
9          int sum = 0;
10         while (records.hasNext()) {
11             element = records.next();
12             PactInteger i =
13                 element.getField(1, PactInteger.class);
14             sum += i.getValue();
15         }
16
17         this.theInteger.setValue(sum);
18         element.setField(1, this.theInteger);
19         out.collect(element);
20     }
```

Word Count Example

18

```
1  public static class WordCountOutFormat extends
    FileOutputStream {
2  private final StringBuilder buffer = new StringBuilder();
3
4  @Override
5  public void writeRecord(PactRecord record) throws
    IOException {
6      this.buffer.setLength(0);
7      this.buffer.append(
8          record.getField(0, PactString.class));
9      this.buffer.append(' ');
10     this.buffer.append(
11         record.getField(1, PactInteger.class).getValue());
12     this.buffer.append('\n');
13
14     byte[] bytes = this.buffer.toString().getBytes();
15     this.stream.write(bytes);
16 }
17 }
```

Composite Keys

19

- Reduce, CoGroup, and Match use keys
- May be composed of more than one field
- Useful for block identifier: separate row and column

Composite Keys

- Reduce, CoGroup, and Match use keys
- May be composed of more than one field
- Useful for block identifier: separate row and column
- Assuming a block is a three-tuple (row, column, data):

```
1 ReduceContract reducer = new ReduceContract(MergeBlocks.  
    class,  
2    new Class[] { PactInteger.class, PactInteger.class },  
    // <-- key classes  
3    new int[] { 0, 1 }, // <-- key indices  
4    mapper); // <-- input
```

Comparison to Key/Value-Pairs

20

- Key/value-pairs easier to understand
- May use generic syntax to check compatibility
- Implicit type specification through generics

Comparison to Key/Value-Pairs

20

- Key/value-pairs easier to understand
- May use generic syntax to check compatibility
- Implicit type specification through generics

- Tuple-based is more flexible, especially for optimization
- Pays quickly off in more complex tasks
- More convention based, field semantic less clear
- UDF more verbose, especially when reusing objects
⇒ Work on class mapping, HLL uses schema inference
- Generalization of k/v-pairs, may simulate k/v

Outline

21

- 1 Overview over Stratosphere
- 2 Dataflow Orientation
- 3 Tuple-based Data Model
- 4 Other Differences**
- 5 Seminar Organization

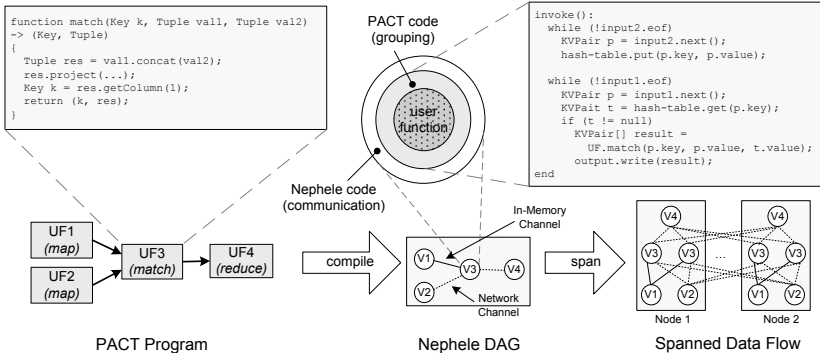
Separate Execution Engine

- Nephele and Pact together are equivalent to Hadoop
- Nepehle may be used by itself for fine-grain data management
- However, decoupling may loose some optimization potential

Separate Execution Engine

22

- Nephelē and Pact together are equivalent to Hadoop
- Nephelē may be used by itself for fine-grain data management
- However, decoupling may lose some optimization potential
- UDF is wrapped in Pact and Nephelē code



Dynamic Resource Allocation

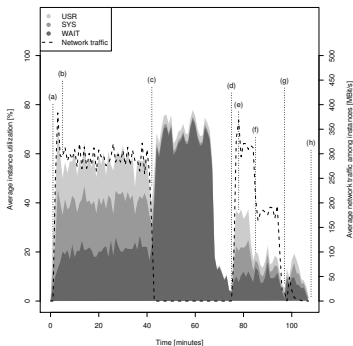
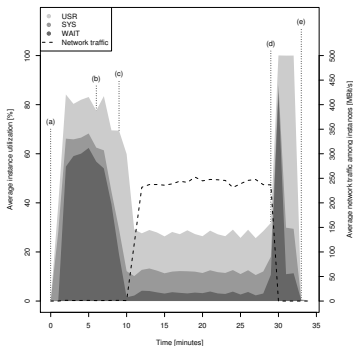
23

- Hadoop works best on cluster environment
- Stratosphere also targets a true cloud environment
- Computation units may be dynamically booked and released

Dynamic Resource Allocation

23

- Hadoop works best on cluster environment
- Stratosphere also targets a true cloud environment
- Computation units may be dynamically booked and released
- Aggregate the smallest 20% numbers, Nephele (l), Hadoop (r)



Continuous Reoptimization

24

- Stratosphere wants to optimize plans
- However, environment (cloud or cluster) is not stable
- An apriori optimal plan may be bad after a while

Continuous Reoptimization

24

- Stratosphere wants to optimize plans
- However, environment (cloud or cluster) is not stable
- An apriori optimal plan may be bad after a while

- Start with **robust** initial plan
- Adapt and optimize during **runtime**

Smart Checkpointing

25

- (Work in Progress)
- Hadoop materializes all results
- Very good for fault-tolerance
- A crashed computation unit may be immediately replaced

Smart Checkpointing

25

- (Work in Progress)
- Hadoop materializes all results
- Very good for fault-tolerance
- A crashed computation unit may be immediately replaced
- However, often bad for runtime performance

- Stratosphere materializes only data when meaningful
- Mostly, when materialization is cheaper than recalculation
 - Will materialize result of computation-intensive tasks
 - Will not materialize cartesian products

Outline

26

- 1 Overview over Stratosphere
- 2 Dataflow Orientation
- 3 Tuple-based Data Model
- 4 Other Differences
- 5 Seminar Organization**

Get Stratosphere Running

27

- `https://www.stratosphere.eu/register/register`
- **Write me email to be unlocked for stage1**
- **Install git and maven2**
- `mkdir/cd stratosphere`
- `git clone https://stratosphere.eu/git/stage1.git .`
- `git checkout -b version02 remotes/origin/version02`
- `mvn install`
- `http://www.stratosphere.eu/projects/Stratosphere/wiki/GettingStarted`
- **Start local mode**
- **Get word count running**

Port Hadoop to Stratosphere

28

- Meeting on 01/10/2012
- Conceptual port should be completed
- Implementation started
- Write email when troubled
- Write tickets if bug was found
- Help other teams, write to sdaa mailinglist

- Who uses joins, composite keys?

Cluster Times

29

- Proposal: for the next 6 weeks, every group gets a fixed day
- Thursday 6pm – Monday 6pm
- Time slots are flexibly tradable
- Announce start/end on cluster mailinglist

Final presentations

30

- May be rescheduled if everybody agrees
- First part should be similar to Hadoop talk
- Second part should include detailed benchmark and comparison
- State what you will additionally evaluate until report

Report

31

- Two column format (sig-alternate.cls)
- 6–8 pages, max. 2 appendix
- How did you change the original algorithms?
- Which design decisions did you make?
- Focus on important, interesting evaluations
- What results did you get? Are they making sense?
- How would you improve the algorithm for better runtime/results?
- No code, only conceptual