Big Data looks Tiny from the Stratosphere

Volker Markl

http://www.user.tu-berlin.de/marklv
volker.markl@tu-berlin.de
Data and analyses are becoming increasingly complex!

Data

- Size (volume)
- Freshness (velocity)
- Format/Media Type (variability)
- Uncertainty/Quality (veracity)
- etc.

Analysis

- Selection/Grouping (map/reduce)
- Relational Operators (Join/Correlation)
- Extraction & Integration, ML, Optimization (map/reduce or dataflow systems) (R, S+, Matlab)
- Predictive Models (map/reduce) (R, S+, Matlab)
- etc.
Overview of Big Data Systems

- Query/Scripting Language:
  - SQL
  - XQuery

- High-level API:
  - Dremel
  - Dataflow Processor

- Compiler/Optimizer:
  - SQL
  - XQuery

- Execution Engine:
  - Dremel
  - Pregel

- Data Store:
  - Bigtable
  - GFS
  - HDFS
  - GPFS-SNC

- Resource Management:
  - YARN
  - Mesos

- Stacks:
  - IBM DB2 / Oracle / Exadata / Teradata
  - Aster / Greenplum
  - Google
  - Apache
  - IBM InfoSphere BigInsights
  - Microsoft
  - Stratosphere
  - ASTERIX
  - BDAS

- Other:
  - column store++
  - a query plan
  - scalable parallel sort

- SQL--
  - XQuery?
Outline

- Stratosphere architecture
  - Layered and flexible stack for massively parallel data management

- The PACT programming model
  - Using second-order functions for data parallelism

- Stratosphere optimizer
  - Deeply embedding Map/Reduce Style UDFs into a query optimizer

- Iterative algorithms in Stratosphere
  - Teaching a dataflow system to execute iterative algorithms with comparable performance to specialized systems
The Stratosphere System Stack

Layered approach – several entry points to the system

PACT Program

- Meteor Script
- Pact 4 Scala

- SOPREMO Compiler
- Scala-Compiler Plugin

- Stratosphere Optimizer
- Runtime Operators
- Nephele Dataflow Engine
The Stratosphere System Stack

Nephele parallel dataflow engine
- Resource allocation
- Scheduling
- Task communication
- Fault tolerance
- Execution monitoring
The Stratosphere System Stack

Runtime engine
- Memory management
- Asynchronous IO
- Query execution (sorting, hashing, …)

Stratosphere Optimizer

Runtime Operators

Nephele Dataflow Engine

PACT Program

Meteor Script

Pact 4 Scala

SOPREMO Compiler

Scala-Compiler Plugin

Nephele Parallel Dataflow
The Stratosphere System Stack

Stratosphere optimizer picks:
- Physical execution strategies
- Partitioning strategies
- Operator order

PACT Program

- Meteor Script
- Pact 4 Scala

SOPREMO Compiler

Scala-Compiler Plugin

Stratosphere Optimizer

Runtime Operators

Nephele Dataflow Engine

Nephele Parallel Dataflow
The PACT programming model

Second-order functions for data parallelism
Parallelization Contracts (PACTs)

- Generalize Map/Reduce
- Describe how input is partitioned/grouped as second order function
  - “What is processed together?”
- First-order UDF called once per input group
- Map PACT (record at a time, 1-dimensional)
  - Each input record forms a group
  - Each record is independently processed by UDF
- Reduce PACT (set at a time, 1-dimensional)
  - One attribute is the designated key
  - All records with same key value form a group
More Parallelization Contracts

Cross PACT
Each pair of input records forms a group
Distributed Cartesian product
Record-at-a-time, 2-dimensional

CoGroup PACT
All pairs with equal key values form a group
2D Reduce
Set-at-a-time, 2-dimensional

Match PACT
Each pair with equal key values forms a group
Distributed equi-join

More PACTs currently under consideration
- For similarity operators, stream processing, etc
A PACT program is an *arbitrary dataflow DAG* consisting of operators.

An operator consists of:
- A *second-order function* (SOF) signature (PACT)
- A user-defined first-order function (FOF) written in Java

PACT programs serve as intermediate representation, but are also exposed to the user:
- To implement UDFs for functionality not supported by Meteor.
The Stratosphere Optimizer

Opening the Black Boxes (VLDB 2012)
Optimizer Design

- Cost-based optimizer produces physical execution plan given PACT program
  - Annotates edges with distribution patterns, e.g., broadcast, partition
  - Chooses physical execution strategies (e.g., hash/sort)
  - Reorders PACT functions
  - Constructs “Nephele job graph”

- **Challenge:** Semantics of user-defined functions unknown
  - How to derive correct transformations (this talk)
  - How to cost functions (ongoing work)
  - Mix and match UDFs and native operators (ongoing work)
Optimization Overview

- **Approach:**
  - Statically analyze user code in each PACT UDFs and extract properties
  - Based on these properties, derive semantically correct transformations
  - Enumerate semantically equivalent plans

- **Contribution:** How to *deeply embed* MapReduce functions into a query optimizer
  - Parallelization and reordering
  - Applies to data flows composed (in part) of functions written in arbitrary imperative code
  - Exportable to Scope, SQL/MapReduce (e.g., Aster, Greenplum)
... via Static Code Analysis

```java
void match (Record left,
            Record right,
            Collector col) {
    Record out = copy (left);
    if (left.get(0) > 3) {
        double a = right.get(2);
        out.set(2,1.0/a);
    }
    out.set(1, 42);
    out.set(3,right.get(0));
    out.set(4,right.get(1));
    out.set(5,right.get(2));
    col.emit (out);
}
```

Feasible:
1. Record data model, fixed API for
2. No control flow between operators

Correct:
- Difficulty comes from different code paths
- Correctness guaranteed through conservatism
- Add to $R,W$ when in doubt
Opening the Black Boxes …

Analyze user code to discover:

- **Output schema** $O_f$: Schema of output record given schema of input record(s)
- **Read set** $R_f$: Attributes of the input record(s) that might influence output
- **Write set** $W_f$: Attributes of the output record(s) that might have different values from respective input attributes
- **Emit cardinality** $E_f$: Bounds on records emitted per call (1, >1, …)

$PACT(f) = (O_f, R_f, W_f, E_f)$
Code Analysis Algorithm

- $R_f$ from get statements
- $W_f$ by backwards traversal of data flow graph starting from emit statement
- $E_f$ by traversing control flow graph

Input 1 = [A,B,C]
Input 2 = [D,E,F]
Output = [A,B,C,D,E,F]

$R_f$ = {A,B,C,D,E,F}
$W_f$ = {B,C}
$E_f$ = 1

```java
1 void match (Record left, Record right, Collector col) {
2     Record out = copy (left);
3     if (left.get(0) > 3) {
4         double a = right.get(2);
5         out.set(2, 1.0/a);
6     }
7     out.set(1, 42);
8     out.set(3, right.get(0));
9     out.set(4, right.get(1));
10    out.set(5, right.get(2));
11    col.emit (out);
12 }
```
Automatic Parallelization

- Optimizer can pick partitioning strategies
  - From PACT signature
- E.g., for Match: broadcast, partition, SFR
- Partitioning strategies propagated top-down as interesting properties
- Can infer preserved partitioning via R/W sets
  - Identifies *pass-through UDFs*
- A Reduce does not always imply a physical sort operator
Operator Reordering

- Reordering PACTs
  - Reduce data volume
  - Introduce new partitioning opportunities
- Reordering, partitioning, and physical operators in one stage
  - “Optimal” execution plan
- Powerful transformations using *read and write conflicts*
- Can “emulate” most relational optimizations without knowing operator semantics
Example Transformations

**Theorem 1:** Two Map operators can be reordered if their UDFs have only read-read conflicts

**Theorem 2:** For a Map and a Reduce, we need in addition the Reduce key groups to be preserved

Enabled optimizations:
- Selection push-down
- (Bushy) join reordering
- Aggregation push-down
- Equivalent to invariant grouping transformation [Chaudhuri & Shim 1994]
- Reordering of non-relational Reduce functions

In VLDB’12 paper: Formal proofs and conditions for safe reorderings for all possible PACT pairs based on $R_f, W_f, E_f$
Support for Iterative queries

Spinning Fast Iterative Dataflows (VLDB 2012)
"Bulk" Iterations

- Recompute state at each iteration
- Conceptual feedback edge in the dataflow – lazy unrolling possible
- Distinguish dynamic data path (different data each iteration) and constant data path (same)
  - Caching heuristics were constant and dynamic paths meet
  - Cached data may be indexed
- Optimizer weighs costs for constant and dynamic data path differently
  - Automatically favors plans that push work to the constant path
PageRank: Two Optimizer Plans

1. Reduce (on tid) (pid=tid, r=\sum k)
   - Sum up partial ranks
   - \textit{fifo}
   - Match (on pid) (tid, k=r*p)
   - Join P and A
   - \textit{CA\textbf{C}E}
   - \textit{buildHashTable (pid)}
   - broadcast
   - P (pid, r)

2. Reduce (on tid) (pid=tid, r=\sum k)
   - Sum up partial ranks
   - \textit{part./sort (tid)}
   - Match (on pid) (tid, k=r*p)
   - Join P and A
   - \textit{CA\textbf{C}E}
   - \textit{buildHashTable (pid)}
   - probeHashTable (pid)
   - partition (pid)
   - P (pid, r)
Sparse Computational Dependencies

- Parts of the state change at each iteration, based on the parts that changed at the previous iteration
  - Most graph algorithms and beyond
  - Huge savings if we do not recompute the whole state

- Need in-place updates in persistent state while surfacing a pure functional model

![Graph Diagram](image-url)
“Incremental” Iterations

- Different programming abstraction based on workset algorithms
- Algorithm works on two sets: Solution Set and Workset
- A delta to the solution set is computed from the workset
- A new workset is recomputed at each iteration
- Solution set is efficiently merged with delta set

```plaintext
function FIXPOINT(f, s)
    while s < f(s) do
        s = f(s)
    return s
```

```plaintext
function INCR(δ, u, S, W)
    while W ≠ ∅ do
        D ← u(S, W)
        W ← δ(D, S, W)
        S ← S ∪ D
    return S
```
Pregel as an Extended PACT Plan

Working Set has messages sent by the vertices

Create Messages from new state

Graph Topology

$W_{i+1}$

$D_{i+10}$

Match

CoGroup

$W_i$

$S_i$

Delta set has state of changed vertices

Aggregate messages and derive new state

In-place updates in persistent hash table
Big Data looks Tiny from the Stratosphere

Stratosphere is a declarative, massively-parallel Big Data Analytics System, funded by DFG and EIT, available open-source under the Apache license.

Data analysis program

```java
$employees = read 'employees.json';
$result = transform $emp in $employees into {
  taxes: $emp.brutto - $emp.netto
  address: {
    $emp.address.*,
    country: 'Germany'
  }
};
write $result to 'output.json';
```

PACT Dataflow

Program compiler

Stratosphere optimizer

Automatic selection of parallelization, shipping and local strategies, operator order and placement

Execution plan

Runtime operators

Hash- and sort-based out-of-core operator implementations, memory management

Job graph

Parallel execution engine

Task scheduling, network data transfers, resource allocation, checkpointing

http://www.stratosphere.eu