Cloud Computing (IN4392)

D.H.J. Epema and A. Iosup. With contributions from Claudio Martella and Bogdan Ghit.

The images used in this lecture courtesy of many anonymous contributors via Google Images. Many other sources were also used. They all retain credit and rights. We thank them for the material.
Terms for Today’s Discussion

Programming model

= language + libraries + runtime system that create a model of computation (an abstract machine)
= “an abstraction of a computer system” Wikipedia

Examples: message-passing vs shared memory, data- vs task-parallelism, ...

Abstraction level

= distance from physical machine

Examples: Assembly low-level vs Java is high level

Many design trade-offs: performance, ease-of-use, common-task optimization, programming paradigm, ...

Q: What is the best abstraction level?
Characteristics of a Cloud Programming Model

1. Cost model (Efficiency) = cost/performance, overheads, ...
2. Scalability, e.g., energy proportional, sub-linear in items...
3. Fault-tolerance, e.g., can tolerate at most 1 failure, ...
4. Support for specific services, e.g., replication, ...
5. Control model, e.g., fine-grained many-task scheduling
6. Data model, including partitioning and placement, out-of-memory data access, etc.
7. Synchronization model, e.g., super-steps, ...
Today’s Challenges

- eScience
- The Fourth Paradigm
- The Data Deluge and Big Data
- Possibly others
eScience: The Why

• Science experiments already cost 25—50% budget
  • ... and perhaps incur 75% of the delays

• Millions of lines of code with similar functionality
  • Little code reuse across projects and application domains
  • ... but last two decades’ science is very similar in structure

• Most results difficult to share and reuse
  • Case-in-point: Sloan Digital Sky Survey
digital map of 25% of the sky × spectra
40TB+ sky survey data
200M+ astro-objects (images)
1M+ objects with spectrum (spectra)
How to make it work for this and the next generation of scientists?


- **A new scientific method**
  - Combine science with IT
  - Full scientific process: control scientific instrument or produce data from simulations, gather and reduce data, analyze and model results, visualize results
  - Mostly compute-intensive, e.g., simulation of complex phenomena

- **IT support**
  - Infrastructure: LHC Grid, Open Science Grid, DAS, NorduGrid, ...
  - From programming models to infrastructure management tools

- **Examples**
  - * physics, Bioinformatics, Material science, Engineering, CompSci

*Q: Why is CompSci an example here?*
The Fourth Paradigm: The Why (An Anecdotal Example)

The Overwhelming Growth of Knowledge

“When 12 men founded the Royal Society in 1660, it was possible for an educated person to encompass all of scientific knowledge. [...] In the last 50 years, such has been the pace of scientific advance that even the best scientists cannot keep up with discoveries at frontiers outside their own field.”

Tony Blair, PM Speech, May 2002

<table>
<thead>
<tr>
<th>Number of Publications</th>
<th>1993</th>
<th>1997</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>1,248,733</td>
<td>1,265,808</td>
</tr>
<tr>
<td>EU15 (net total)</td>
<td>1,180,730</td>
<td>1,347,985</td>
</tr>
<tr>
<td>United Kingdom</td>
<td>309,683</td>
<td>342,535</td>
</tr>
<tr>
<td>Germany</td>
<td>268,393</td>
<td>318,286</td>
</tr>
<tr>
<td>Japan</td>
<td>289,751</td>
<td>336,858</td>
</tr>
<tr>
<td>France</td>
<td>203,814</td>
<td>232,058</td>
</tr>
<tr>
<td>Canada</td>
<td>168,331</td>
<td>166,216</td>
</tr>
<tr>
<td>Italy</td>
<td>122,398</td>
<td>147,023</td>
</tr>
<tr>
<td>Switzerland</td>
<td>57,664</td>
<td>66,761</td>
</tr>
<tr>
<td>Netherlands</td>
<td>83,600</td>
<td>92,526</td>
</tr>
</tbody>
</table>

Data: King, The scientific impact of nations, Nature’04.
The Fourth Paradigm: The What

From Hypothesis to Data

- Thousand years ago: science was **empirical** describing natural phenomena
- Last few hundred years: **theoretical** branch using models, generalizations
- Last few decades: a **computational** branch simulating complex phenomena
- Today (**the Fourth Paradigm**): data exploration unify theory, experiment, and simulation
  - Data captured by instrument
  - Processed by software
  - Information/Knowledge
  - Scientist analyzes results using data management and statistics

Q1: What is the Fourth Paradigm?

Q2: What are the dangers of the Fourth Paradigm?

Source: Jim Gray and “The Fourth Paradigm”,
"Everywhere you look, the quantity of information in the world is soaring. According to one estimate, mankind created 150 exabytes (billion gigabytes) of data in 2005. This year, it will create 1,200 exabytes. Merely keeping up with this flood, and storing the bits that might be useful, is difficult enough. Analysing it, to spot patterns and extract useful information, is harder still."

The Data Deluge, The Economist, 25 February 2010
“Data Deluge”: The Personal Memex Example

- Vannevar Bush in the 1940s: record your life
- MIT Media Laboratory: The Human Speechome Project/TotalRecall, data mining/analysis/visualization
  - Deb Roy and Rupal Patel “record practically every waking moment of their son’s first three years” (20% privacy time...Is this even legal?! Should it be?!)
  - 11x1MP/14fps cameras, 14x16b-48KHz mics, 4.4TB RAID + tapes, 10 computers; 200k hours audio-video
  - Data size: 200GB/day, 1.5PB total
“Data Deluge”: Radioastronomy

Will generate TerraBytes of data for each observation
- Observation = 1 night
- Stations = hundreds
- Time-step <= 1s

SKA = Square Kilometer Array
- Hundreds to thousands of times bigger
What Happens in an Internet Minute? [In 2013]

- 639,800 GB of global IP data transferred
- 20 New victims of identity theft
- 204 million Emails sent
- 47,000 App downloads
- 61,141 Hours of music
- 20 million Photo views
- 320+ New Twitter accounts
- 277,000 Logins
- 100,000 New tweets
- 6 million Facebook views
- 2+ million Search queries
- 30 Hours of video uploaded
- 1.3 million Video views

And Future Growth is Staggering

Data at the Core of Our Society:
The LinkedIn Example

The State of LinkedIn

Sources: Vincenzo Cosenza, The State of LinkedIn, [http://vincos.it/the-state-of-linkedino](http://vincos.it/the-state-of-linkedino/)
“Data Deluge” In Summary

Data Deluge = data generated by humans and devices (IoT)

- Interacting
- Understanding
- Deciding
- Creating

Sources: IDC, EMC.
The Challenge: The Three “V”s of Big Data
When You Can, Keep and Process Everything
* New queries later

- **Volume**
  - More data vs. better models
  - Exponential growth + iterative models
  - Scalable storage and distributed queries

- **Velocity**
  - Speed of the feedback loop
  - Gain competitive advantage: fast recommendations
  - Analysis in near-real time to extract value

- **Variety**
  - The data can become messy: text, video, audio, etc.
  - Difficult to integrate into applications

Too big, too fast, does not comply with traditional DB

Vs of big data

- Volume – large scale of data
- Velocity – timeliness of data
- Variety – different forms of data
- Veracity – uncertainty of data
- **Vicissitude** – dynamic combination of several big data Vs in processing systems that support the addition of new queries at run-time
The “Big Cake” in the Data Center

Online Social Networks

= HPC/MapReduce/other framework

Universe Explorers

Multiple frameworks = Isolation, especially performance

Financial Analysts

Enthusiast
Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)
   1. Bags of Tasks
   2. Workflows
   3. Parallel Programming Models
4. Programming Models for Big Data
5. Summary
What is a Bag of Tasks (BoT)? A System View

BoT = set of jobs sent by a user...

\[ W_u = \{ J_i | user(J_i) = u \} \]

...that should complete together. Often, start ~with first job.

\[ ST(J') \leq ST(J) + \Delta \]

• Why Bag of Tasks? From the perspective of the user, jobs in set are just tasks of a larger job
• A single useful result from the complete BoT
• Result can be combination of all tasks, or a selection of the results of most or even a single task

Q: What is the user’s view?

Applications of the BoT Programming Model

- Parameter sweeps
  - Comprehensive, possibly exhaustive investigation of a model
  - Very useful in engineering and simulation-based science

- Monte Carlo simulations
  - Simulation with random elements: fixed time yet limited inaccuracy
  - Very useful in engineering and simulation-based science

- Many other types of batch processing
  - Periodic computation, Cycle scavenging
  - Very useful to automate operations and reduce waste
BoTs Became the Dominant Programming Model for Grid Computing

From jobs [%]

From CPUMTime [%]

Iosup and Epema: Grid Computing Workloads.
Practical Applications of the BoT Programming Model
Parameter Sweeps in Condor [1/4]

• Sue the scientist wants to “Find the value of F(x,y,z) for 10 values for x and y, and 6 values for z”

• **Solution**: Run a parameter sweep, with 10 x 10 x 6 = 600 parameter values

• **Problem of the solution**:
  • Sue runs one job (a combination of x, y, and z) on her low-end machine. It takes 6 hours.
  • That’s **150 days** uninterrupted computation on Sue’s machine!

Parameter Sweeps in Condor [2/4]

Universe = vanilla
Executable = sim.exe
Input = input.txt
Output = output.txt
Error = error.txt
Log = sim.log

Requirements = OpSys == “WINNT61” &&
Arch == “INTEL” &&
(Disk >= DiskUsage) && ((Memory * 1024) >= ImageSize)

InitialDir = run_$($Process)
Queue 600

Complex SLAs can be specified easily

Also passed as parameter to sim.exe

Source: Condor Team, Condor User’s Tutorial.
http://cs.uwisc.edu/condor
Practical Applications of the BoT Programming Model
Parameter Sweeps in Condor [3/4]

% condor_submit sim.submit
Submitting job(s)

................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................
................................................

Logging submit event(s)

................................................
................................................
................................................
................................................
................................................
................................................
................................................

600 job(s) submitted to cluster 3.

Source: Condor Team, Condor User’s Tutorial.
http://cs.uwisc.edu/condor
Practical Applications of the BoT Programming Model

Parameter Sweeps in Condor [4/4]

% condor_q
-- Submitter: x.cs.wisc.edu : <128.105.121.53:510>:
x.cs.wisc.edu

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0</td>
<td>frieda</td>
<td>4/20 12:08</td>
<td>0+00:00:05</td>
<td>R</td>
<td>0</td>
<td>9.8</td>
<td>sim.exe</td>
</tr>
<tr>
<td>3.1</td>
<td>frieda</td>
<td>4/20 12:08</td>
<td>0+00:00:03</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>sim.exe</td>
</tr>
<tr>
<td>3.2</td>
<td>frieda</td>
<td>4/20 12:08</td>
<td>0+00:00:01</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>sim.exe</td>
</tr>
<tr>
<td>3.3</td>
<td>frieda</td>
<td>4/20 12:08</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>sim.exe</td>
</tr>
</tbody>
</table>

...

| 3.598| frieda| 4/20 12:08| 0+00:00:00 | I  | 0   | 9.8  | sim.exe |
| 3.599| frieda| 4/20 12:08| 0+00:00:00 | I  | 0   | 9.8  | sim.exe |

600 jobs; 599 idle, 1 running, 0 held

Source: Condor Team, Condor User’s Tutorial.
http://cs.uwisc.edu/condor
Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)
3. **Programming Models for Compute-Intensive Workloads**
   1. Bags of Tasks
   2. Workflows
   3. Parallel Programming Models
4. Programming Models for Big Data
5. Summary
What is a Workflow?

WF = set of jobs with precedence (think Direct Acyclic Graph)
Applications of the Workflow Programming Model

• Complex applications
  • Complex filtering of data
  • Complex analysis of instrument measurements

• Applications created by non-CS scientists*
  • Workflows have a natural correspondence in the real-world, as descriptions of a scientific procedure
  • Visual model of a graph sometimes easier to program

• Precursor of the MapReduce Programming Model (next slides)

Workflows Existed in Grids, but Did Not Become a Dominant Programming Model

- Traces
  - T1: DEE 09/06-10/07 4,113 122k 152
  - T2: EE2 05/07-11/07 1,030 46k 41

- Selected Findings
  - Loose coupling
  - Graph with 3-4 levels
  - Average WF size is 30/44 jobs
  - 75%+ WFs are sized 40 jobs or less, 95% are sized 200 jobs or less

Practical Applications of the WF Programming Model
Bioinformatics in Taverna

Source: Carole Goble and David de Roure, Chapter in “The Fourth Paradigm”,
Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)

   1. Bags of Tasks
   2. Workflows

3. Parallel Programming Models

4. Programming Models for Big Data

5. Summary
Parallel Programming Models

- Abstract machines
  - (Distributed) shared memory
  - Distributed memory: MPI

- Conceptual programming models
  - Master/worker
  - Divide and conquer
  - Data / Task parallelism
  - BSP

- System-level programming models
  - Threads on GPUs and other multi-cores
Compute-Intensive Workloads in the Cloud

- BoTs
- Workflows
- HPC
- Many-thread
- Etc.

**Task** (groups of 5, 5 minutes): discuss parallel programming in clouds

**Task** (inter-group discussion): discuss.
Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)
4. Programming Models for Big Data
5. Summary
Ecosystems of Big-Data Programming Models

Q: Where does MR-on-demand fit? Q: Where does Pregel-on-GPUs fit?

Flume BigQuery → SQL → Meteor → JAQL → Hive → Pig → Sawzall

PACT MapReduce Model

Flume Engine → Dremel Service Tree → Tera Data Engine → Azure Data Engine

Nephele

Hadoop/ Giraph YARN

HDFS → Voldemort

MAPREDUCE PROGRAMMING MODEL

MPI/ Erlang → Dryad

Hydraxs

AQL

DryadLINQ

High-Level Language

Scope

Execution Engine

S3

GFS

Tera Data Store

Azure Data Store

Storage Engine

LFS

CosmosFS

Asterix B-tree

Voldemort

HDFS

Q: Where does MR-on-demand fit? Q: Where does Pregel-on-GPUs fit?

* Plus Zookeeper, CDN, etc.

Adapted from: Dagstuhl Seminar on Information Management in the Cloud, http://www.dagstuhl.de/program/calendar/partlist/?semnr=11321&SUOG
Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)
4. Programming Models for Big Data
   1. MapReduce
   2. Graph Processing
   3. Other Big Data Programming Models
5. Summary
MapReduce

- Model for processing and generating large data sets
- Enables a functional-like programming model
- Splits computations into independent parallel tasks
- Makes efficient use of large commodity clusters
- Hides the details of parallelization, fault-tolerance, data distribution, monitoring and load balancing
MapReduce: The Programming Model

A programming **model**, not a programming **language**!

1. **Input/Output:**
   - Set of key/value pairs

2. **Map Phase:**
   - Processes input key/value pair
   - Produces set of intermediate pairs
   
   \[
   \text{map (in\_key, in\_value)} \rightarrow \text{list(out\_key, interm\_value)}
   \]

3. **Reduce Phase:**
   - Combines all intermediate values for a given key
   - Produces a set of merged output values

   \[
   \text{reduce(out\_key, list(interm\_value))} \rightarrow \text{list(out\_value)}
   \]
Colored Square Counter

![Diagram of colored square counter process]

2011-2012
MapReduce in the Cloud

- **Facebook use case:**
  - Social-networking services
  - Analyze connections in the graph of friendships to recommend new connections

- **Google use case:**
  - Web-base email services, GoogleDocs
  - Analyze messages and user behavior to optimize ad selection and placement

- **Youtube use case:**
  - Video-sharing sites
  - Analyze user preferences to give better stream suggestions
More Examples of MapReduce Applications

- Distributed Grep
- Count of URL Access Frequency
- Reverse Web-Link Graph
- Term-Vector per Host
- Distributed Sort
- Inverted indices
- Page ranking
- Machine learning algorithms
- ...
Q: What is the **performance problem** raised by this step?
MapReduce Workloads in the Cloud

- BoTs
- Workflows
- HPC
- Many-thread
- Etc.

**Task** (groups of 5, 5 minutes): discuss parallel programming in clouds

**Task** (inter-group discussion): discuss.
The BTWorld Workflow of MapReduce Jobs

Dynamic Big Data Processing
Fawkes = Elastic MapReduce via Two-level scheduling architecture

Elastic MapReduce

MapReduce framework
- Distributed file system
- Execution engine
- Data locality constraints

Because workloads may be time-varying:
- Poor resource utilization
- Imbalanced service levels

Grow and shrink MapReduce
- High resource utilization
- Reconfiguration for balanced service levels
- Break data locality

No data locality

Core nodes

- Classical deployment
- Uniform data distribution
- No removal

Transient nodes (TR)

- No local storage
- R/W from/to core nodes
- Instant removal

Performance?

Relaxed data locality

Core nodes

- Classical deployment
- Uniform data distribution
- No removal

Trans-core nodes (TC)

- Local storage, no input
- Only R from core nodes
- Delayed removal

Better performance?

FAWKES in a nutshell

1. Size of MapReduce cluster
   - Changes dynamically
   - Balanced by weight
     \[ S_i = \frac{W_i}{\sum W_j} \]

2. Updates dynamic weights when
   - New frameworks arrive
   - Framework states change

3. Shrinks and grows frameworks to
   - Allocate new frameworks (min. shares)
   - Give fair shares to existing ones

Performance of dynamic MapReduce

- TR - good for compute-intensive workloads.
- TC - needed for disk-intensive workloads.

Dynamic MapReduce: < 25% overhead

Performance of FAWKES

<table>
<thead>
<tr>
<th>Nodes</th>
<th>45</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frameworks</td>
<td>3</td>
</tr>
<tr>
<td>Min. shares</td>
<td>10</td>
</tr>
<tr>
<td>Datasets</td>
<td>300 GB</td>
</tr>
<tr>
<td>Jobs submitted</td>
<td>900</td>
</tr>
</tbody>
</table>

Up to 20% lower slowdown

None – Minimum shares
EQ – EQual shares
TD – Task Demand
PU – Processor Usage
JS – Job Slowdown

FAWKES: behind the scenes

EQ

Utilizations: 60% / 23% / 5%

Imbalanced

TD

Utilizations: 50% / 30% / 8%

More balanced

Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)

4. Programming Models for Big Data
   1. MapReduce
   2. Graph Processing
   3. Other Big Data Programming Models

5. Summary
Graph Processing Example
Single-Source Shortest Path (SSSP)

- Dijkstra’s algorithm
  - Select node with minimal distance
  - Update neighbors
  - $O(|E| + |V| \cdot \log|V|)$ with FiboHeap

- Initial dataset:
  A: <0, (B, 5), (D, 3)>
  B: <inf, (E, 1)>
  C: <inf, (F, 5)>
  D: <inf, (B, 1), (C, 3), (E, 4), (F, 4)>
  ...

Source: Claudio Martella, Presentation on Giraph at TU Delft, Apr 2012.
Graph Processing Example
SSSP in MapReduce

Q: What is the performance problem?

- Mapper output: distances 
  \(<A, \langle 0, (B, 5), (D, 3) \rangle > \ldots \)

- But also graph structure 
  \(<A, \langle 0, (B, 5), (D, 3) \rangle > \ldots \)

- Reducer input: distances 
  B: \(<\text{inf}, 5\rangle, D: \(<\text{inf}, 3\rangle \ldots \)

- But also graph structure 
  B: \(<\text{inf}, (E, 1)\rangle \ldots \)

N jobs, where N is the graph diameter

Source: Claudio Martella, Presentation on Giraph at TU Delft, Apr 2012.
The Pregel Programming Model for Graph Processing

- Batch-oriented processing
- Runs in-memory
- Vertex-centric API
- Fault-tolerant
- Runs on Master-Slave architecture

- OK, the actual model follows in the next slides

Source: Claudio Martella, Presentation on Giraph at TU Delft, Apr 2012.
The Pregel Programming Model for Graph Processing

Based on Valiant’s

Bulk Synchronous Parallel (BSP)

- N processing units with fast local memory
- Shared communication medium
- Series of **Supersteps**
- Global Synchronization Barrier
- Ends when all voteToHalt

- Pregel executes initialization, one or several supersteps, shutdown

Source: Claudio Martella, Presentation on Giraph at TU Delft, Apr 2012.
Pregel
The Superstep

• Each Vertex (execution in parallel)
  • Receive messages from other vertices
  • Perform own computation (user-defined function)
  • Modify own state or state of outgoing messages
  • Mutate topology of the graph
  • Send messages to other vertices

• Termination condition
  • All vertices inactive
  • All messages have been transmitted

Source: Pregel article.
Pregel: The Vertex-Based API

```cpp
template <typename VertexValue, 
          typename EdgeValue, 
          typename MessageValue>

class Vertex {
  public:
    virtual void Compute(MessageIterator* msgs) = 0;

    const string& vertex_id() const;
    int64 superstep() const;

    const VertexValue& GetValue();
    VertexValue* MutableValue();
    OutEdgeIterator GetOutEdgeIterator();

    void SendMessageTo(const string& dest_vertex,
                        const MessageValue& message);
    void VoteToHalt();
};
```

Source: Pregel article.
The Pregel Architecture
Master-Worker

- Master assigns vertices to Workers
  - Graph partitioning
- Master coordinates Supersteps
- Master coordinates Checkpoints
- Workers execute vertices `compute()`
- Workers exchange messages directly

Source: Pregel article.
Apache Giraph
An Open-Source Implementation of Pregel

- Loose implementation of Pregel
- Strong community (Facebook, Twitter, LinkedIn)
- Runs 100% on existing Hadoop clusters
- Single Map-only job

2012-2013

http://incubator.apache.org/giraph/
What is the performance of these platforms?

- **Graph500**
  - Single application (BFS), Single class of synthetic datasets

- Few existing platform-centric comparative studies
  - Prove the superiority of a given system, limited set of metrics

Our vision: a benchmarking suite for graph processing across all platforms
Our Method

A benchmark suite for performance evaluation of graph-processing platforms

1. **Multiple Metrics**, e.g.,
   - Execution time
   - Normalized: EPS, VPS
   - Utilization

2. **Representative graphs** with various characteristics, e.g.,
   - Size
   - Directivity
   - Density

3. **Typical graph algorithms**, e.g.,
   - BFS
   - Connected components

---


Platforms we have evaluated

- Distributed or non-distributed
- Graph-specific or generic

Experimental setup

• Size
  • Most experiments take 20 working nodes
  • Up to 50 working nodes

• DAS4: a multi-cluster Dutch grid/cloud
  • Intel Xeon 2.4 GHz CPU (dual quad-core, 12 MB cache)
  • Memory 24 GB
  • 10 Gbit/s Infiniband network and 1 Gbit/s Ethernet network
  • Utilization monitoring: Ganglia

• HDFS used here as distributed file system
BFS: results for all platforms, all data sets

- No platform runs fastest for every graph
- Not all platforms can process all graphs
- Hadoop is the worst performer


Giraph: results for all algorithms, all data sets

- Storing the whole graph in memory helps Giraph perform well
- Giraph may crash when graphs or number of messages large


Key Findings From the Study of 6 Platforms

- Performance is function of (Dataset, Algorithm+Data Structure, Platform, Deployment)
  - Previous performance studies may lead to tunnel vision
  - Also looked at data structure, for CPU/GPU

- Platforms have their own drawbacks (crashes, long execution time, tuning, etc.)

- Some platforms can scale up reasonably with cluster size (horizontally) or number of cores (vertically)
  - Strong vs weak scaling still a challenge—workload scaling tricky

Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)
4. Programming Models for Big Data
   1. MapReduce
   2. Graph Processing
   3. Other Big Data Programming Models
5. Summary
Stratosphere/Apache Flink (Aug 2014)

- **Meteor** query language, **Supremo** operator framework

- Programming Contracts (PACTs) programming model
  - Extended set of 2\textsuperscript{nd} order functions (vs MapReduce)
  - Declarative definition of data parallelism

- **Nephele** execution engine
  - Schedules multiple dataflows simultaneously
  - Supports IaaS environments based on Amazon EC2, Eucalyptus

- **HDFS** storage engine
Stratosphere
Programming Contracts (PACTs) [1/2]

Parallelization Contract (PACT)

Key
Value

Input Data

2nd-order function

Input Contract (Reduce)

User Code
First-order function

Independent Data Subsets

Output Data

MAP

REDUCE

CROSS

MATCH

Also in MapReduce

Source: PACT overview,
Stratosphere Programming Contracts (PACTs) [2/2]

Q: How can PACTs optimize data processing?

Stratosphere
Nephele

Client

Public Network (Internet)

JobManager

Private / Virtualized Network

Task Manager

Task Manager

Task Manager

Task Manager

Persistent Storage
Stratosphere vs MapReduce

- PACT extends MapReduce
  - Both propose 2\textsuperscript{nd}-order functions (5 PACTs vs Map & Reduce)
  - Both require from user 1\textsuperscript{st}-order functions (what’s inside the Map)
  - Both can benefit from higher-level languages
  - PACT ecosystem has IaaS support

- Key-value data model

Key:
- Used to build independent subsets
- Must be comparable and hashable
- Does not need to be unique
  - no Primary Key semantic!
- Interpreted only by user code

Value:
- Holds application data
- Interpreted only by user code
- Often struct-like data type to hold multiple values

Source: Fabian Hueske, Large Scale Data Analysis Beyond MapReduce, Hadoop Get Together, Feb 2012.
Stratosphere vs MapReduce
Triangle Enumeration [1/3]

- Input: undirected graph edge-based
- Output: triangles

1. Read input graph
2. Sort by smaller vertex ID
3. Join edges to triads
4. Triads to triangles

Source: Fabian Hueske, Large Scale Data Analysis Beyond MapReduce, Hadoop Get Together, Feb 2012 and Stratosphere example.
Stratosphere vs MapReduce Triangle Enumeration [2/3]

MapReduce Formulation

1st job computes triads

2nd job closes triads

Edges are scanned twice

Awkward join implementation

Data is sorted twice

Potentially huge Intermediate result is materialized in HDFS

Source: Fabian Hueske, Large Scale Data Analysis Beyond MapReduce, Hadoop Get Together, Feb 2012 and Stratosphere example.
Stratosphere vs MapReduce
Triangle Enumeration [3/3]

Stratosphere Formulation

First part of pipeline identical to MapReduce

Single scan of edges

Intermediate result is streamed through hash-table

Edge data is used to build hash-table for join while triads are built

Source: Fabian Hueske, Large Scale Data Analysis Beyond MapReduce, Hadoop Get Together, Feb 2012 and Stratosphere example.
Agenda

1. Introduction
2. Cloud Programming in Practice (The Problem)
4. Programming Models for Big Data
5. Summary
Conclusion Take-Home Message

• Programming model = computer system abstraction

• Programming Models for Compute-Intensive Workloads
  • Many trade-offs, few dominant programming models
  • Models: bags of tasks, workflows, master/worker, BSP, ...

• Programming Models for Big Data
  • Big data programming models have ecosystems
  • Many trade-offs, many programming models
  • Models: MapReduce, Pregel, PACT, Dryad, ...
  • Execution engines: Hadoop, Koala+MR, Giraph, PACT/Nephele, Dryad, ...

• Reality check: cloud programming is maturing
Reading Material (Really Active Field)

- **Workloads**

- **The Fourth Paradigm**

- **Programming Models for Compute-Intensive Workloads**

- **Programming Models for Big Data**
  - Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150
  - Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein, Khaled Elmeleegy, Russell Sears: MapReduce Online. NSDI 2010: 313-328
  - Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski: Pregel: a system for large-scale graph processing. SIGMOD Conference 2010: 135-146
  - Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, Daniel Warneke: Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. SoCC 2010: 119-130