ASCI-A24 Cloud Computing
Cloud Programming Models

D.H.J. Epema and A. Iosup. With contributions from Claudio Martella and Bogdan Ghit.
2012-2013

Parallel and Distributed Group/TU Delft
Terms for Today’s Discussion

Programming model

= language + libraries + runtime system 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

Q: What is the best 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, ...
Characteristics of a Cloud Programming Model

1. Cost model (Efficiency) = cost/performance, overheads, ...
2. Scalability (elasticity)
3. Fault-tolerance
4. Support for specific services
5. Control model, e.g., fine-grained many-task scheduling
6. Data access model, including partitioning and placement, out-of-memory data access, etc.
7. Synchronization model
Agenda

1. Introduction

2. **Cloud Programming in Practice (The Problem)**


4. Programming Models for Big Data

5. Summary
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 x 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?

Source: Jim Gray and Alex Szalay, “eScience -- A Transformed Scientific Method”,

• **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><strong>Netherlands</strong></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 instruments
  - 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?

"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
What is “Big Data”? 

- Very large, distributed aggregations of loosely structured data, often incomplete and inaccessible 
- Easily exceeds the processing capacity of conventional database systems 
- Principle of Big Data: "When you can, keep everything!" 
- Too big, too fast, and doesn’t comply with the traditional database architectures
The Three “V”s of Big Data

• **Volume**
  - More data vs. better models
  - Data grows exponentially
  - Analysis in near-real time to extract value
  - Scalable storage and distributed queries

• **Velocity**
  - Speed of the feedback loop
  - Gain competitive advantage: fast recommendations
  - Identify fraud, predict customer churn faster

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

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 start at most \( \Delta \)s after the 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

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

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 $10 \times 10 \times 6 = 600$ 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!

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

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.
Practical Applications of the BoT Programming Model
Parameter Sweeps in Condor [4/4]

% condor_q

```
% 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>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.598</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>
<tr>
<td>3.599</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>
```

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)
   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

<table>
<thead>
<tr>
<th>Trace</th>
<th>Source</th>
<th>Duration</th>
<th>Number of WFs</th>
<th>Number of Tasks</th>
<th>CPUdays</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>DEE</td>
<td>09/06-10/07</td>
<td>4,113</td>
<td>122k</td>
<td>152</td>
</tr>
<tr>
<td>T2</td>
<td>EE2</td>
<td>05/07-11/07</td>
<td>1,030</td>
<td>46k</td>
<td>41</td>
</tr>
</tbody>
</table>

- 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
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?

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: 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
     \[
     \text{reduce(out\_key, list(interm\_value))} \rightarrow \text{list(out\_value)}
     \]
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
Wordcount Example

• File 1: “The big data is big.”
• File 2: “MapReduce tames big data.”

• Map Output:
  • Mapper-1: (The, 1), (big, 1), (data, 1), (is, 1), (big, 1)
  • Mapper-2: (MapReduce, 1), (tames, 1), (big, 1), (data, 1)

• Reduce Input
  • Reducer-1: (The, 1)
  • Reducer-2: (big, 1), (big, 1), (big, 1)
  • Reducer-3: (data, 1), (data, 1)
  • Reducer-4: (is, 1)
  • Reducer-5: (MapReduce, 1), (MapReduce, 1)
  • Reducer-6: (tames, 1)

• Reduce Output
  • Reducer-1: (The, 1)
  • Reducer-2: (big, 3)
  • Reducer-3: (data, 2)
  • Reducer-4: (is, 1)
  • Reducer-5: (MapReduce, 2)
  • Reducer-6: (tames, 1)
Colored Square Counter

2011-2012
Q: What is the performance problem raised by this step?
What is Hadoop? A MapReduce Exec. Engine

- Inspired by Google, supported by Yahoo!

- Designed to perform fast and reliable analysis of the big data

- Large expansion in many domains such as:
  - Finance, technology, telecom, media, entertainment, government, research institutions
MapReduce Evolution [1/5]
Hadoop is Maturing: Important Contributors

Lifetime patches contributed for all Hadoop related projects: community members by current employer

Source: [http://www.theregister.co.uk/2012/08/17/community_hadoop/](http://www.theregister.co.uk/2012/08/17/community_hadoop/)
KOALA and MapReduce [1/2]

- KOALA
  - Placement & allocation
  - Central for all MR clusters
  - Maintains MR cluster metadata
- On-demand MR clusters
  - Performance isolation
  - Data isolation
  - Failure isolation
  - Version isolation

- MR-Runner
  - Configuration & deployment
  - MR cluster monitoring
  - Grow/Shrink mechanism

Koala and MapReduce [2/2]

Performance of the Resizing Mechanism

- Stream of 50 MR jobs
- MR cluster of 20 core nodes + 20 transient nodes
- GGP increases the size of the data transferred across the network
- GSP grows/shrinks based on the resource utilization of the cluster
- GGDP enables local writes on the disks of provisioned nodes

F_{min} = 0.25
F_{max} = 1.25
growStep = 5
shrinkStep = 2
T_{GSP} = 30 s
T_{GG(D)P} = 120 s

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: $<\text{inf}, (E, 1)>$
  C: $<\text{inf}, (F, 5)>$
  D: $<\text{inf}, (B, 1), (C, 3), (E, 4), (F, 4)>$
  ...
Graph Processing Example
SSSP in MapReduce

Q: What is the performance problem?

- Mapper output: distances
  \( <B, 5>, <D, 3>, <C, \text{inf}>, \ldots \)
- But also graph structure
  \( <A, <0, (B, 5), (D, 3)>, \ldots \)

- Reducer input: distances
  B: \( \langle \text{inf}, 5 \rangle, \text{D: } <\text{inf}, 3 \rangle, \ldots \)
- But also graph structure
  B: \( <\text{inf}, (E, 1)>, \ldots \)

\( N \) 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 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.
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

Implement processing algorithm

```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.
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

Persistent computation state

Zookeeper

NN & JT

Tasktracker

Map Slot

Tasktracker

Map Slot

Tasktracker

Map Slot

Tasktracker

Map Slot

Master

2012-2013

http://incubator.apache.org/giraph/
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

• 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

Input Contract (Reduce)

User Code

First-order function

Independent Data Subsets

Output Data

2\textsuperscript{nd}-order function

Also in MapReduce

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

Q: How can PACTs optimize data processing?
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.
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

2012-2013