Cloud Computing (IN4392)

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

2012-2013
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

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
3. Fault-tolerance
4. Support for specific services
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
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?
“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>
<th>2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States</td>
<td>1,248,733</td>
<td>1,265,808</td>
<td></td>
</tr>
<tr>
<td>EU15 (net total)</td>
<td>1,180,730</td>
<td>1,347,985</td>
<td></td>
</tr>
<tr>
<td>United Kingdom</td>
<td>309,683</td>
<td>342,535</td>
<td></td>
</tr>
<tr>
<td>Germany</td>
<td>268,393</td>
<td>318,286</td>
<td></td>
</tr>
<tr>
<td>Japan</td>
<td>289,751</td>
<td>336,858</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>203,814</td>
<td>232,058</td>
<td></td>
</tr>
<tr>
<td>Canada</td>
<td>168,331</td>
<td>166,216</td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td>122,398</td>
<td>147,023</td>
<td></td>
</tr>
<tr>
<td>Switzerland</td>
<td>57,664</td>
<td>66,761</td>
<td></td>
</tr>
<tr>
<td>Netherlands</td>
<td>83,600</td>
<td>92,526</td>
<td></td>
</tr>
</tbody>
</table>

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?

The “Data Deluge”: The Why

"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/visio
  - 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”: The Gaming Analytics Example

- EQ II: 20TB/year all logs
- Halo3: 1.4PB served statistics on player logs
“Data Deluge”: Datasets in Comp.Sci.

Peer-to-Peer Trace Archive
... PWA, ITA, CRAWDAD, ...

- 1,000s of scientists: From theory to practice
"Data Deluge":
The Professional World Gets Connected

The State of LinkedIn

150,000,000
registered members

Feb 2012

Source: Vincenzo Cosenza, The State of LinkedIn, http://vincos.it/the-state-of-linkedin/
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

## Data Warehouse vs. Big Data

**Traditional Data Warehouse**
- Complete record from transactional system
- All data centralized
- Addition every month/day of new data
- Analytics designed against stable environment
- Many reports run on a production basis

**Big-data Analytic Environment**
- Data from many sources inside and outside of organization, including traditional DW
- Data often physically distributed
- Need to iterate solution to test/improve models
- Large-memory analytics also part of iteration
- Every iteration usually requires complete reload of information

Source: [http://wikibon.org/](http://wikibon.org/)
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 Bag of Tasks (BoT)? A System View

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

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

...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

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!

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]

```bash
% 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>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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)
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**
  - | Trace | Source | Duration  | Number of WFs | Number of Tasks | CPUdays |
  - |------|--------|-----------|---------------|----------------|---------|
  - | 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

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

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

- System-level programming models
  - Threads on GPUs and other multi-cores

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

Task (inter-group discussion): discuss.

2012-2013

Varbanescu et al.: Towards an Effective Unified Programming Model for Many-Cores. IPDPS WS 2012
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/?semlr=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
   
   \[
   \text{map} \ (\text{in}\_\text{key}, \ \text{in}\_\text{value}) \rightarrow \text{list}(\text{out}\_\text{key}, \ \text{interm}\_\text{value})
   \]

3. Reduce Phase:
   - Combines all intermediate values for a given key
   - Produces a set of merged output values
   
   \[
   \text{reduce}(\text{out}\_\text{key}, \ \text{list}(\text{interm}\_\text{value})) \rightarrow \text{list}(\text{out}\_\text{value})
   \]
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

split

map

shuffle/sort

reduce

2011-2012
MapReduce @ Google: Example 1

- Generating language model statistics
  - Count # of times every 5-word sequence occurs in large corpus of documents (and keep all those where count $\geq 4$)

- MapReduce solution:
  - Map: extract 5-word sequences $\Rightarrow$ count from document
  - Reduce: combine counts, and keep if count large enough

http://www.slideshare.net/jhammerb/mapreduce-pact06-keynote
MapReduce @ Google: Example 2

- Joining with other data
  - Generate per-doc summary, but include per-host summary (e.g., # of pages on host, important terms on host)

- MapReduce solution:
  - Map: extract host name from URL, lookup per-host info, combine with per-doc data and emit
  - Reduce: identity function (emit key/value directly)

http://www.slideshare.net/jhammerb/mapreduce-pact06-keynote
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
- ...

2011-2012
Q: What is the performance problem raised by this step?
Execution Overview (2/2)

- One master, many workers example
  - Input data split into M map tasks (64 / 128 MB): 200,000 maps
  - Reduce phase partitioned into R reduce tasks: 4,000 reduces
  - Dynamically assign tasks to workers: 2,000 workers

- Master assigns each map / reduce task to an idle worker
  - Map worker:
    - Data locality awareness
    - Read input and generate R local files with key/value pairs
  - Reduce worker:
    - Read intermediate output from mappers
    - Sort and reduce to produce the output
Failures and Back-up Tasks

(a) Normal execution  
(b) No backup tasks  
(c) 200 tasks killed

Data transfer rates over time for different executions of the sort program
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
Hadoop @ Yahoo

- When you visit yahoo, you are interacting with data processed with Hadoop!

Hadoop Use Cases

1. Search
   • Yahoo, Amazon, Zvents

2. Log processing
   • Facebook, Yahoo, Joost, Last.fm

3. Recommendation Systems
   • Facebook

4. Data Warehouse
   • Facebook, AOL

5. Video and Image Analysis
   • New York Times, Eyealike

• http://cloud.berkeley.edu/data/hdfs.pdf
Hadoop Distributed File System

- Assumptions and goals:
  - Data distributed across hundreds or thousands of machines
  - Detection of faults and automatic recovery
  - Designed for batch processing vs. interactive use
  - High throughput of data access vs. low latency of data access
  - High aggregate bandwidth and high scalability
  - Write-once-read-many file access model
  - Moving computations is cheaper than moving data: minimize network congestion and increase throughput of the system
HDFS Architecture

- **Master/slave architecture**

- **NameNode (NN)**
  - Manages the file system namespace
  - Regulates access to files by clients
  - Open/close/rename files or directories
  - Mapping of blocks to DataNodes

- **DataNode (DN)**
  - One per node in the cluster
  - Manages local storage of the node
  - Block creation/deletion/replication initiated by NN
  - Serve read/write requests from clients
HDFS Internals

• Replica Placement Policy
  • First replica on one node in the local rack
  • Second replica on a different node in the local rack
  • Third replica on a different node in a different rack
  ⇒ improved write performance (2/3 are on the local rack)
  ⇒ preserves data reliability and read performance

• Communication Protocols
  • Layered on top of TCP/IP
  • Client Protocol: client – NameNode machine
  • DataNode Protocol: DataNodes – NameNode
  • NameNode responds to RPC requests issued by DataNodes / clients
Hadoop Scheduler

• Job divided into several independent tasks executed in parallel
  • The input file is split into chunks of 64 / 128 MB
  • Each chunk is assigned to a map task
  • Reduce task aggregate the output of the map tasks

• The master assigns tasks to the workers in FIFO order
  • JobTracker: maintains a queue of running jobs, the states of the TaskTrackers, the tasks assignments
  • TaskTrackers: report their state via a heartbeat mechanism

• Data Locality: execute tasks close to their data
• Speculative execution: re-launch slow tasks
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/
MapReduce Evolution [2/5]

Late Scheduler

- LATE Scheduler
  - Longest Approximate Time to End
  - Speculatively execute the task that will finish farthest into the future
  - \( \text{time\_left} = \frac{(1 - \text{progressScore})}{\text{progressRate}} \)
  - Tasks make progress at a roughly constant rate
  - Robust to node heterogeneity

- EC2 Sort running times
  - LATE vs. Hadoop vs. No spec.

MapReduce Evolution [3/5]

FAIR Scheduling

- Isolation and statistical multiplexing
- Two-level architecture:
  - First, allocates task slots across pools
  - Second, each pool allocates its slots among multiple jobs
- Based on a max-min fairness policy

Figure 3: Slot allocation example. Figure (a) shows pool demands (as boxes) and minimum shares (as dashed lines). The algorithm proceeds in three phases: fill buckets whose minimum share is more than their demand (b), fill remaining buckets up to their minimum share (c), and distribute remaining slots starting at the emptiest bucket (d).

MapReduce Evolution [4/5]

Delay Scheduling

- Data locality issues:
  - Head-of-line scheduling: FIFO, FAIR
  - Low probability for small jobs to achieve data locality
  - 58% of jobs @ FACEBOOK have < 25 maps
  - Only 5% achieve node locality vs. 59% rack locality

MapReduce Evolution [5/5]

Delay Scheduling

• Solution:
  • Skip the head-of-line job until you find a job that can launch a local task
  • Wait T1 seconds before launching tasks on the same rack
  • Wait T2 seconds before launching tasks off-rack
  • T1 = T2 = 15 seconds => 80% data locality

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3 maps</td>
<td>2% / 50%</td>
<td>75% / 96%</td>
</tr>
<tr>
<td>10 maps</td>
<td>37% / 98%</td>
<td>99% / 100%</td>
</tr>
<tr>
<td>100 maps</td>
<td>84% / 99%</td>
<td>94% / 99%</td>
</tr>
</tbody>
</table>

Table 3: Locality in small-jobs experiment.

KOALA and MapReduce [1/9]

- **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

![Diagram of KOALA and MR-Runner integration](image)
Koala and MapReduce [2/9]

Resizing Mechanism

• Two types of nodes:
  • Core nodes: fully-functional nodes, with TaskTracker and DataNode (local disk access)
  • Transient nodes: compute nodes, with TaskTracker

• Parameters
  • \( F = \) Number of running tasks per number of available slots
  • Predefined \( F_{\text{min}} \) and \( F_{\text{max}} \) thresholds
  • Predefined constant step \( \text{growStep} / \text{shrinkStep} \)
  • \( T = \) time elapsed between two successive resource offers

• Three policies:
  • Grow-Shrink Policy (GSP): grow-shrink but maintain \( F \) between \( F_{\text{min}} \) and \( F_{\text{max}} \)
  • Greedy-Grow Policy (GGP): grow, shrink when workload done
  • Greedy-Grow-with-Data Policy (GGDP): GGP with core nodes (local disk access)
Koala and MapReduce [3/9]

Workloads

- 98% of jobs @ Facebook process 6.9 MB and take less than a minute [1]

- Google reported in 2004 computations with TB of data on 1000s of machines [2]

---


Wordcount (Type 0, full system)

- 100 GB input data
- 10 core nodes with 8 map slots each
- 800 map tasks executed in 7 waves
- Wordcount is CPU-bound in the map phase
Koala and MapReduce [5/9]

Sort (Type 0, full system)

- Performed by the MR framework during the shuffling phase
- Intermediate key/value pairs are processed in increasing key order
- Short map phase with 40%-60% CPU utilization
- Long reduce phase which is highly disk intensive
a) Wordcount (Type 1)

- Speedup relative to an MR cluster with 10 core nodes
- Close to linear speedup on core nodes

b) Sort (Type 2)
- Input data of 40 GB
- Wordcount output data = 20 KB
- Sort output data = 40 GB
- **Wordcount scales better than Sort on transient nodes**
Koala and MapReduce [8/9]

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

$F_{\text{min}} = 0.25$
$F_{\text{max}} = 1.25$
$\text{growStep} = 5$
$\text{shrinkStep} = 2$
$T_{\text{GSP}} = 30\ s$
$T_{\text{GG}(D)P} = 120\ s$
• **Goal:** Process 10s TB of P2P traces

• **DAS-4 constraints:**
  • Time limit per job (15 min)
  • Non-persistent storage

• **Solution:**
  • Reserve the nodes for several days, import and process the data?

• **Our approach:**
  • Split the data into multiple subsets
  • Smaller data sets => faster import and processing
  • Setup multiple MR clusters, one for each subset
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, inf>, ... 
- But also graph structure
  <A, <0, (B, 5), (D, 3)> ... 
- Reducer input: distances
  B: <inf, 5>, D: <inf, 3> ... 
- But also graph structure
  B: <inf, (E, 1)> ...

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();
};
```

Implements processing algorithm

Input message

Output message

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.
Pregel Performance
SSSP on 1 Billion-Vertex Binary Tree

Source: Pregel article.
Pregel Performance
SSSP on Random Graphs, Various Sizes

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

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

http://incubator.apache.org/giraph/
Page rank benchmarks

- **Tiberium Tan**
  - Almost 4000 nodes, shared among numerous groups in Yahoo!
  - Hadoop 0.20.204 (secure Hadoop)
  - 2x Quad Core 2.4GHz, 24 GB RAM, 1x 6TB HD

- **org.apache.giraph.benchmark.PageRankBenchmark**
  - Generates data, number of edges, number of vertices, # of supersteps
  - 1 master/ZooKeeper
  - 20 supersteps
  - No checkpoints
  - 1 random edge per vertex

Source: Avery Ching presentation at HortonWorks. 
http://www.slideshare.net/averyching/20111014hortonworks/
Worker scalability (250M vertices)

Source: Avery Ching presentation at HortonWorks.  
http://www.slideshare.net/averyching/20111014hortonworks/
Vertex scalability (300 workers)

Source: Avery Ching presentation at HortonWorks.
http://www.slideshare.net/averyching/20111014hortonworks/
Vertex/worker scalability

Source: Avery Ching presentation at HortonWorks. 
http://www.slideshare.net/averyching/20111014hortonworks/
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

MAP

REDUCE

CROSS

MATCH

Also in MapReduce

2012-2013

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

Q: How can PACTs optimize data processing?

Stratosphere Nephele

Diagram of Stratosphere Nephele architecture:
- Client
  - Public Network (Internet)
  - Private / Virtualized Network
    - JobManager
      - 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
Pairwise Shortest Paths [1/3]

- Floyd-Warshall Algorithm
  - For \( k \) from 1 to \( n \)
    - For \( i \) from 1 to \( n \)
      - For \( j \) from 1 to \( n \)
        - \( D_{i,j} \leftarrow \min( D_{i,j}, D_{i,k} + D_{k,j} ) \)

Source: Stratosphere example,
https://stratosphere.eu/wiki/lib/exe/detail.php/wiki:all2all_sp_taskdescription.png?id=wiki%3Aa2aspexample
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

- Single scan of edges
- First part of pipeline identical to MapReduce
- 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
• **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