Before the tutorial starts: Visit https://goo.gl/M5ABf7 for preliminary setup instructions!

Jan. 23rd, 2018, Zagreb, Croatia
About me

Permanent **Research Scientist** at University of Luxembourg

→ Part of the PCOG Team led by Prof. P. Bouvry since 2007

→ **Research interests:**
  - High Performance Computing
  - Security (crash/cheating faults, obfuscation, blockchains)
  - Performance of HPC/Cloud/IoT platforms and services

Manager of the **UL HPC Facility** with Prof. P. Bouvry since 2007

→ ≃ 206.772 TFllops (2017), 7952.4 TB

→ expert UL HPC team (S. Varrette, V. Plugaru, S. Peter, H. Cartiaux, C. Parisot)

National / EU HPC projects:

→ ETP4HPC, EU COST NESUS…

→ PRACE[2] (acting Advisor)

→ EuroHPC / IPCEI on HPC and Big Data (BD) Applications
Welcome!

3\textsuperscript{rd} NESUS WS on Data Science & Heterogeneous Computing

**In this session: Tutorial on Big Data Analytics**

- Focus on **practicals tools** rather than theoretical content
- starts with **daily data management** …
  - … before speaking about **Big** data management
  - in particular: data transfer (over **SSH**), data versioning with **Git**
- continue with **classical tools** and their usage in HPC
  - review HPC environments and the hands-on environment
    - reviewing **Environment Modules** and **Lmod**
    - introducing **Vagrant** and **Easybuild**
  - introduction to Big Data processing engines: **Hadoop**, **Spark**
  - introduction to **Tensorflow**, an ML/DL processing framework
Disclaimer: Acknowledgements

- Part of these slides were **courtesy** borrowed w. permission from:
  - Prof. Martin Theobald *(Big Data and Data Science Research Group), UL*
- Part of the slides material adapted from:
  - Advanced Analytics with Spark, O Reilly
  - Data Analytics with HPC courses
    - © CC AttributionNonCommercial-ShareAlike 4.0
- the hands-on material is adapted from several resources:
  - (of course) the **UL HPC School**, **credits**: UL HPC team
    - S. Varrette, V. Plugaru, S. Peter, H. Cartiaux, C. Parisot
  - similar Github projects:
    - Jonathan Dursi: [hadoop-for-hpcers-tutorial](https://github.com/jdursi/hadoop-for-hpcers-tutorial)
### Agenda: Jan. 23th, 2018

**Lecture & hands-on:**
**Big Data Analytics: Overview and Practical Examples**

http://nesusws-tutorials-BD-DL.rtfd.io

<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00 - 09:30</td>
<td>Discover the Hands-on tool: Vagrant</td>
</tr>
<tr>
<td>09:30 - 10:00</td>
<td>HPC and Big Data (BD): Architectures and Trends</td>
</tr>
<tr>
<td>10:00 - 10:30</td>
<td>Interlude: Software Management in HPC systems</td>
</tr>
<tr>
<td>10:30 - 11:00</td>
<td>[Big] Data Management in HPC Environment: Overview and Challenges</td>
</tr>
<tr>
<td>11:00 - 11:15</td>
<td><strong>Coffee Break</strong></td>
</tr>
<tr>
<td>11:15 - 12:30</td>
<td>Big Data Analytics with Hadoop &amp; Spark</td>
</tr>
<tr>
<td>12:30 - 13:00</td>
<td>Deep Learning Analytics with Tensorflow</td>
</tr>
<tr>
<td>13:00</td>
<td><strong>Lunch</strong></td>
</tr>
</tbody>
</table>
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Introduction

Before we start...
Overview of HPC & BD Trends
Main HPC and DB Components

Interlude: Software Management in HPC systems

[Big] Data Management in HPC Environment: Overview and Challenges
Performance Overview in Data transfer
Data transfer in practice
Sharing Data

Big Data Analytics with Hadoop & Spark
Apache Hadoop
Apache Spark

Deep Learning Analytics with Tensorflow
Introduction
Before we start...
Overview of HPC & BD Trends
Main HPC and DB Components

Interlude: Software Management in HPC systems

[Big] Data Management in HPC Environment: Overview and Challenges
Performance Overview in Data transfer
Data transfer in practice
Sharing Data

Big Data Analytics with Hadoop & Spark
Apache Hadoop
Apache Spark

Deep Learning Analytics with Tensorflow
Follow instructions on Getting Started / Pre-requisites

- create (if needed) accounts: Github, Vagrant Cloud, Docker Hub
- install mandatory software, i.e. (apart from Git):

<table>
<thead>
<tr>
<th>Platform</th>
<th>Software</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS</td>
<td>Homebrew</td>
<td>The missing package manager for macOS</td>
<td>brew install ...</td>
</tr>
<tr>
<td>Mac OS</td>
<td>Brew Cask Plugin</td>
<td>Mac OS Apps install made easy</td>
<td>brew cask install ...</td>
</tr>
<tr>
<td>Mac OS</td>
<td>iTerm2</td>
<td>(optional) enhanced Terminal</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>MobaXTERM</td>
<td>Terminal with tabbed SSH client</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>Git for Windows</td>
<td>may be you guessed...</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>SourceTree</td>
<td>(optional) enhanced git GUI</td>
<td></td>
</tr>
<tr>
<td>Windows/Linux</td>
<td>Virtual Box</td>
<td>Free hypervisor provider for Vagrant</td>
<td></td>
</tr>
<tr>
<td>Windows/Linux</td>
<td>Vagrant</td>
<td>Reproducible environments made easy.</td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td>Docker for Ubuntu</td>
<td>Lightweight Reproducible Containers</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>Docker for Windows</td>
<td>Lightweight Reproducible Containers</td>
<td></td>
</tr>
</tbody>
</table>
Introduction

Discover the Hands-on Tool: Vagrant

Development environments made easy.

Create and configure lightweight, reproducible, and portable development environments.

http://vagrantup.com/
What is Vagrant?

Create and configure lightweight, reproducible, and portable development environments

- **Command line** tool
- Easy and Automatic per-project VM management
  - Supports many hypervisors: VirtualBox, VMWare...
  - Easy text-based configuration (Ruby syntax) Vagrantfile
- Supports **provisioning** through configuration management tools
  - Shell
  - Puppet
  - Salt...

**Cross-platform**: runs on Linux, Windows, MacOS
Installation Notes


- **Mac OS X:**
  - best done using **Homebrew** and **Cask**
  
  `brew install caskroom/cask/brew-cask`
  `brew cask install virtualbox`  # install virtualbox
  `brew cask install vagrant`
  `brew cask install vagrant-manager`  # cf http://vagrantmanager.com/

- **Windows / Linux:**
  - install **Oracle Virtualbox** and the **Extension Pack**
  - install **Vagrant**
Why use Vagrant?

- Create new VMs quickly and easily: only one command!
  - `vagrant up`
- Keep the number of VMs under control
  - All configuration in `VagrantFile`
- **Reproducibility**
  - Identical environment in development and production
- **Portability**
  - Avoid sharing 4 GB VM disks images
  - `Vagrant Cloud` to share your images
- **Collaboration made easy:**
  
  ```bash
  $ git clone ...
  $ vagrant up
  ```
Minimal default setup

```
$> vagrant init [-m] <user>/<name> # setup vagrant cloud image
```

- A Vagrantfile is configured for box `<user>/<name>`
  - Find existing box: Vagrant Cloud [https://vagrantcloud.com/](https://vagrantcloud.com/)
  - You can have multiple (named) box within the same Vagrantfile
    ✓ See ULHPC/puppet-sysadmins/Vagrantfile
    ✓ See Falkor/tutorials-BD-ML/Vagrantfile

```ruby
Vagrant.configure(2) do |config|
  config.vm.box = '<user>/<name>'
  config.ssh.insert_key = false
end
```

<table>
<thead>
<tr>
<th>Box name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ubuntu/trusty64</td>
<td>Ubuntu Server 14.04 LTS</td>
</tr>
<tr>
<td>debian/contrib-jessie64</td>
<td>Vanilla Debian 8 Jessie</td>
</tr>
<tr>
<td>centos/7</td>
<td>CentOS Linux 7 x86_64</td>
</tr>
</tbody>
</table>
Pulling and Running a Vagrant Box

$\texttt{vagrant up} \quad \# \textit{boot the box(es) set in the Vagrantfile}

- Base box is downloaded and stored locally ~/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
  - The VM is booted and (eventually) provisioned
  - Once within the box: /vagrant = directory hosting Vagrantfile
Pulling and Running a Vagrant Box

```bash
$ vagrant up  # boot the box(es) set in the Vagrantfile
```

- Base box is downloaded and stored locally  ~/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
  - The VM is booted and (eventually) provisioned
  - Once within the box: /vagrant = directory hosting Vagrantfile

```
$ vagrant status  # State of the vagrant box(es)
```

Sebastien Varrette (University of Luxembourg)
Pulling and Running a Vagrant Box

```bash
$ vagrant up  # boot the box(es) set in the Vagrantfile
```

- Base box is downloaded and stored locally ~/.vagrant.d/boxes/
- A new VM is created and configured with the base box as template
  - The VM is booted and (eventually) provisioned
  - Once within the box: /vagrant = directory hosting Vagrantfile

```bash
$ vagrant status  # State of the vagrant box(es)
```

```bash
$ vagrant ssh  # connect inside it, CTRL-D to exit
```
Introduction

Stopping Vagrant Box

```bash
$> vagrant { destroy | halt }
# destroy / halt
```

- Once you have finished your work within a *running* box
  - save the state for later with `vagrant halt`
  - reset changes / tests / errors with `vagrant destroy`
  - commit changes by generating a new version of the box
This tutorial heavily relies on **Vagrant**

→ you will need to familiarize with the tool if not yet done

---

**Your Turn!**

**Hands-on 0**


- **Clone** the tutorial repository
- **Basic Usage of Vagrant**

**Step 1**

**Step 2**
Introduction

Before we start...

Overview of HPC & BD Trends
Main HPC and DB Components

Interlude: Software Management in HPC systems

[BIG] Data Management in HPC Environment: Overview and Challenges
Performance Overview in Data transfer
Data transfer in practice
Sharing Data

Big Data Analytics with Hadoop & Spark
Apache Hadoop
Apache Spark

Deep Learning Analytics with Tensorflow
HPC: High Performance Computing
BD: Big Data
Introduction

Why HPC and BD ?

**HPC:** High Performance Computing

**BD:** Big Data

- Essential tools for **Science, Society and Industry**
  - All scientific disciplines are becoming computational today
    - requires very high computing power, handles **huge** volumes of data

- **Industry, SMEs** increasingly relying on HPC
  - to invent innovative solutions
  - ... while reducing cost & decreasing time to market

---

*To out-compete you must out-compute*

Increasing competition, heightened customer expectations and shortening product development cycles are forcing the pace of acceleration across all industries.
Introduction

Why HPC and BD?

HPC: High Performance Computing
BD: Big Data

- Essential tools for Science, Society and Industry
  - All scientific disciplines are becoming computational today
    ✓ requires very high computing power, handles huge volumes of data
  - Industry, SMEs increasingly relying on HPC
    - to invent innovative solutions
    - ... while reducing cost & decreasing time to market

- HPC = global race (strategic priority) - EU takes up the challenge:
  - EuroHPC / IPCEI on HPC and Big Data (BD) Applications

To out-compete you must out-compute
Increasing competition, heightened customer expectations and shortening product development cycles are forcing the pace of acceleration across all industries.
Introduction

New Trends in HPC

- **Continued scaling** of scientific, industrial & financial applications
  \[\rightsquigarrow\ldots\text{ well beyond Exascale}\]
- New trends changing the landscape for HPC
  \[\rightsquigarrow\text{Emergence of Big Data analytics}\]
  \[\rightsquigarrow\text{Emergence of (Hyperscale) Cloud Computing}\]
  \[\rightsquigarrow\text{Data intensive Internet of Things (IoT) applications}\]
  \[\rightsquigarrow\text{Deep learning & cognitive computing paradigms}\]

Special Study

Analysis of the Characteristics and Development Trends of the Next-Generation of Supercomputers in Foreign Countries

Earl C. Joseph, Ph.D.
Robert Sorensen
Steve Conway
Kevin Monroe

[Source: IDC RIKEN report, 2016]
Introduction

Toward Modular Computing

- Aiming at **scalable, flexible HPC infrastructures**
  - *Primary* processing on CPUs and accelerators
    - ✓ **HPC & Extreme Scale Booster** modules
  - *Specialized modules* for:
    - ✓ HTC & I/O intensive workloads;
    - ✓ [Big] Data Analytics & AI

[Source: "Towards Modular Supercomputing: The DEEP and DEEP-ER projects", 2016]
Introduction

Prerequisites: Metrics

- **HPC**: High Performance Computing
- **BD**: Big Data

Main HPC/BD Performance Metrics

- **Computing Capacity**: often measured in **flops** (or **flop/s**)
  - Floating point operations per seconds
  - $\text{GFlops} = 10^9$  $\text{TFlops} = 10^{12}$  $\text{PFlops} = 10^{15}$  $\text{EFlops} = 10^{18}$
Prerequisites: Metrics

- **HPC**: High Performance Computing
- **BD**: Big Data

### Main HPC/BD Performance Metrics

- **Computing Capacity**: often measured in flops (or flop/s)
  - Floating point operations per second (often in DP)
  - \( \text{GFlops} = 10^9 \quad \text{TFlops} = 10^{12} \quad \text{PFlops} = 10^{15} \quad \text{EFlops} = 10^{18} \)

- **Storage Capacity**: measured in multiples of bytes = 8 bits
  - \( \text{GB} = 10^9 \) bytes \quad \text{TB} = 10^{12} \quad \text{PB} = 10^{15} \quad \text{EB} = 10^{18} \)
  - \( \text{GiB} = 1024^3 \) bytes \quad \text{TiB} = 1024^4 \quad \text{PiB} = 1024^5 \quad \text{EiB} = 1024^6 \)

- **Transfer rate** on a medium measured in Mb/s or MB/s

- Other metrics: Sequential vs Random R/W speed, IOPS
Introduction

Before we start...
Overview of HPC & BD Trends
Main HPC and DB Components

Interlude: Software Management in HPC systems

[Big] Data Management in HPC Environment: Overview and Challenges
Performance Overview in Data transfer
Data transfer in practice
Sharing Data

Big Data Analytics with Hadoop & Spark
Apache Hadoop
Apache Spark

Deep Learning Analytics with Tensorflow
HPC Components: [GP]CPU

**CPU**
- Always multi-core
- **Ex:** Intel Core i7-7700K (Jan 2017) \( R_{peak} \approx 268.8 \text{ GFlops (DP)} \)
  \[ \leftrightarrow 4 \text{ cores @ 4.2GHz (14nm, 91W, 1.75 billion transistors)} \]
  \[ \leftrightarrow + \text{ integrated graphics (24 EUs)} \quad R_{peak} \approx +441.6 \text{ GFlops} \]

**GPU / GPGPU**
- Always multi-core, optimized for vector processing
- **Ex:** Nvidia Tesla V100 (Jun 2017) \( R_{peak} \approx 7 \text{ TFlops (DP)} \)
  \[ \leftrightarrow 5120 \text{ cores @ 1.3GHz (12nm, 250W, 21 billion transistors)} \]
  \[ \leftrightarrow \text{ focus on Deep Learning workloads} \quad R_{peak} \approx 112 \text{ TFLOPS (HP)} \]

\[ \approx 100 \text{ Gflops for 130$ (CPU), 214$ (GPU)} \]
HPC Components: Local Memory

Larger, slower and cheaper

- **SSD (SATA3)** R/W: 550 MB/s; 100000 IOPS
- **HDD (SATA3 @ 7,2 krpm)** R/W: 227 MB/s; 85 IOPS
**Introduction**

**HPC Components: Interconnect**

- **latency**: time to send a minimal (0 byte) message from A to B
- **bandwidth**: max amount of data communicated per unit of time

<table>
<thead>
<tr>
<th>Technology</th>
<th>Effective Bandwidth</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit Ethernet</td>
<td>1 Gb/s</td>
<td>125 MB/s</td>
</tr>
<tr>
<td>10 Gigabit Ethernet</td>
<td>10 Gb/s</td>
<td>1.25 GB/s</td>
</tr>
<tr>
<td>Infiniband QDR</td>
<td>40 Gb/s</td>
<td>5 GB/s</td>
</tr>
<tr>
<td>Infiniband EDR</td>
<td>100 Gb/s</td>
<td>12.5 GB/s</td>
</tr>
<tr>
<td>100 Gigabit Ethernet</td>
<td>100 Gb/s</td>
<td>1.25 GB/s</td>
</tr>
<tr>
<td>Intel Omnipath</td>
<td>100 Gb/s</td>
<td>12.5 GB/s</td>
</tr>
</tbody>
</table>

[Source: www.top500.org, Nov. 2017]

Sebastien Varrette (University of Luxembourg)
**Introduction**

**HPC Components: Interconnect**

- **latency**: time to send a minimal (0 byte) message from A to B
- **bandwidth**: max amount of data communicated per unit of time

<table>
<thead>
<tr>
<th>Technology</th>
<th>Effective Bandwidth</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gigabit Ethernet</td>
<td>1 Gb/s</td>
<td>125 MB/s</td>
</tr>
<tr>
<td>10 Gigabit Ethernet</td>
<td>10 Gb/s</td>
<td>1.25 GB/s</td>
</tr>
<tr>
<td>Infiniband QDR</td>
<td>40 Gb/s</td>
<td>5 GB/s</td>
</tr>
<tr>
<td>Infiniband EDR</td>
<td>100 Gb/s</td>
<td>12.5 GB/s</td>
</tr>
<tr>
<td>100 Gigabit Ethernet</td>
<td>100 Gb/s</td>
<td>1.25 GB/s</td>
</tr>
<tr>
<td>Intel Omnipath</td>
<td>100 Gb/s</td>
<td>12.5 GB/s</td>
</tr>
</tbody>
</table>

[Source: www.top500.org, Nov. 2017]
Network Topologies

**Direct vs. Indirect** interconnect

- *Direct*: each network node attaches to at least one compute node
- *Indirect*: compute nodes attached at the edge of the network only
  - Many routers only connect to other routers.
Network Topologies

**Direct vs. Indirect interconnect**

- **direct**: each network node attaches to at least one compute node
- **indirect**: compute nodes attached at the edge of the network only
  - ✓ many routers only connect to other routers.

**Main HPC Topologies**

- **CLOS Network / Fat-Trees** [Indirect]
  - can be fully non-blocking (1:1) or blocking (x:1)
  - typically enables **best performance**
    - ✓ Non blocking bandwidth, lowest network latency
Introduction

Network Topologies

- **Direct** vs. **Indirect** interconnect
  - *direct*: each network node attaches to at least one compute node
  - *indirect*: compute nodes attached at the edge of the network only
    - many routers only connect to other routers.

Main HPC Topologies

- **CLOS Network / Fat-Trees** [Indirect]
  - can be fully non-blocking (1:1) or blocking (x:1)
  - typically enables **best performance**
    - Non blocking bandwidth, lowest network latency

- **Mesh or 3D-torus** [Direct]
  - Blocking network, cost-effective for systems at scale
  - Great performance solutions for applications with locality
  - Simple expansion for future growth
HPC Components: Operating System

- Exclusively Linux-based (really 100%)
- Reasons:
  - stability
  - prone to devals

[Source: www.top500.org, Nov 2017]
[Big]Data Management

Storage architectural classes & I/O layers

- **DAS**
  - SATA
  - SAS
  - Fiber Channel

- **SAN**
  - SATA
  - SAS
  - Fiber Channel

- **NAS**
  - SATA
  - SAS
  - Fiber Channel

- **Application**

- **[Distributed] File system**
  - Network
    - iSCSI
    - ...
Introduction

[Big]Data Management: Disk Encl.

- $\approx 120 \ K\€$ - enclosure - 48-60 disks (4U)
  - incl. redundant (i.e. 2) RAID controllers (master/slave)
File System (FS)

- Logical manner to store, organize, manipulate & access data
File System (FS)

- Logical manner to **store**, **organize**, **manipulate** & **access** data

- (local) **Disk FS**: FAT32, NTFS, HFS+, ext{3,4}, {x,z,btr}fs...
  - manage data on permanent storage devices
  - *poor* perf. **read**: 100 → 400 MB/s | **write**: 10 → 200 MB/s
Networked FS: NFS, CIFS/SMB, AFP

Disk access from remote nodes via network access

Poorer performance for HPC jobs especially parallel I/O

- **Read**: only 381 MB/s on a system capable of 740 MB/s (16 tasks)
- **Write**: only 90 MB/s on system capable of 400 MB/s (4 tasks)

**Source**: LISA’09 Ray Paden: *How to Build a Petabyte Sized Storage System*

**COMMENT**: Traditionally, a single NFS/CIFS file server manages both user data and metadata operations which “gates” performance/scaling and presents a single point of failure risk. Products (e.g., CNFS) are available that provide multiple server designs to avoid this issue.
[Big]Data Management: File Systems

- **Networked FS**: NFS, CIFS/SMB, AFP
  - disk access from remote nodes via network access
  - poorer performance for HPC jobs especially parallel I/O
    - **read**: only 381 MB/s on a system capable of 740MB/s (16 tasks)
    - **write**: only 90MB/s on system capable of 400MB/s (4 tasks)

  ![Networked File System Diagram]

- **[scale-out] NAS**
  - aka Appliances OneFS...
  - Focus on CIFS, NFS
  - Integrated HW/SW
  - **Ex**: EMC (Isilon), IBM (SONAS), DDN...

**COMMENT:**
Traditionally, a single NFS/CIFS file server manages both user data and metadata operations which "gates" performance-scaling and presents a single point of failure risk. Products (e.g., CNFS) are available that provide multiple server designs to avoid this issue.

[Source : LISA’09] Ray Paden: *How to Build a Petabyte Sized Storage System*
Introduction

[Big]Data Management: File Systems

- **Basic Clustered FS**
  - File access is parallel
  - File System overhead operations is distributed and done in parallel
    - **✓ no** metadata servers
  - File clients access file data through file servers via the LAN

File system overhead operations are *distributed* across the entire cluster and is done in parallel; it is *not* concentrated in any given place. There is no single server bottleneck. User data and metadata flows between all nodes and all disks via the file servers.
Multi-Component Clustered FS

- File access is parallel
- File System overhead operations on dedicated components
  - ✓ metadata server (Lustre) or director blades (Panasas)
- Multi-component architecture
- File clients access file data through file servers via the LAN

**Lustre**

- File client
- File server
- Metadata servers
- LAN
- SAN Fabric
- Storage Controller

**Panasas**

- File client
- File server
- Director blade
- Disks
- File server
- Storage controller
- File server
- Storage controller
- Director blade
- Metadata server
- LAN
[Big]Data Management: FS Summary

- **File System (FS):** Logical manner to store, organize & access data
  - (local) **Disk FS:** FAT32, NTFS, HFS+, ext4, {x,z,btr}fs...
  - **Networked FS:** NFS, CIFS/SMB, AFP
  - **Parallel/Distributed FS:** SpectrumScale/GPFS, Lustre
    - typical FS for HPC / HTC (High Throughput Computing)

<table>
<thead>
<tr>
<th>Type</th>
<th>Read [GB/s]</th>
<th>Write [GB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk FS</td>
<td>0.426</td>
<td>0.212</td>
</tr>
<tr>
<td>Networked FS</td>
<td>0.381</td>
<td>0.090</td>
</tr>
<tr>
<td>Parallel/Distributed FS: SpectrumScale/GPFS, Lustre</td>
<td>10.14</td>
<td>8.41</td>
</tr>
<tr>
<td>Lustre</td>
<td>4.5</td>
<td>2.956</td>
</tr>
</tbody>
</table>

* maximum random read/write, per IOZone or IOR measures, using 15 concurrent nodes for networked FS.
**File System (FS):** Logical manner to *store, organize & access* data

- **(local) Disk FS:** FAT32, NTFS, HFS+, ext4, \{x,z,btr\}fs...
- **Networked FS:** NFS, CIFS/SMB, AFP
- **Parallel/Distributed FS:** SpectrumScale/GPFS, Lustre

✓ typical FS for HPC / HTC (High Throughput Computing)

**Main Characteristic of Parallel/Distributed File Systems**

**Capacity and Performance** increase with \#servers
[Big]Data Management: FS Summary

**File System (FS):** Logical manner to store, organize & access data

- (local) **Disk FS**: FAT32, NTFS, HFS+, ext4, {x,z,btr}fs...
- **Networked FS**: NFS, CIFS/SMB, AFP
- **Parallel/Distributed FS**: SpectrumScale/GPFS, Lustre
  ✓ typical FS for HPC / HTC (High Throughput Computing)

### Main Characteristic of Parallel/Distributed File Systems

**Capacity and Performance** increase with #servers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Read* [GB/s]</th>
<th>Write* [GB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext4</td>
<td>Disk FS</td>
<td>0.426</td>
<td>0.212</td>
</tr>
<tr>
<td>nfs</td>
<td>Networked FS</td>
<td>0.381</td>
<td>0.090</td>
</tr>
<tr>
<td>gpfs (iris)</td>
<td>Parallel/Distributed FS</td>
<td><strong>10.14</strong></td>
<td><strong>8.41</strong></td>
</tr>
<tr>
<td>gpfs (gaia)</td>
<td>Parallel/Distributed FS</td>
<td>7.74</td>
<td>6.524</td>
</tr>
<tr>
<td>lustre</td>
<td>Parallel/Distributed FS</td>
<td>4.5</td>
<td>2.956</td>
</tr>
</tbody>
</table>

* maximum **random** read/write, per IOZone or IOR measures, using 15 concurrent nodes for networked FS.
Introduction

HPC Components: Data Center

Definition (Data Center)

- Facility to house computer systems and associated components
  - Basic storage component: rack (height: 42 RU)

Challenges:
- Power (UPS, battery)
- Cooling
- Fire protection
- Security

Power/Heat dissipation per rack:
- HPC computing racks: 30-120 kW
- Storage racks: kW
- Interconnect racks: kW

Various Cooling Technology
- Airflow
- Direct-Liquid Cooling, Immersion...

Power Usage Effectiveness

PUE = Total facility power
IT equipment power

Sebastien Varrette (University of Luxembourg)
**Definition (Data Center)**

- Facility to house computer systems and associated components
  - Basic storage component: rack (height: 42 RU)

**Challenges:** Power (UPS, battery), Cooling, Fire protection, Security

- Power/Heat dissipation per rack:
  - HPC computing racks: 30-120 kW
  - Storage racks: 15 kW
  - Interconnect racks: 5 kW

**Power Usage Effectiveness**

\[
PUE = \frac{\text{Total facility power}}{\text{IT equipment power}}
\]

- Various Cooling Technology
  - Airflow
  - Direct-Liquid Cooling, Immersion...
Interlude: Software Management in HPC systems

Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow

Sebastien Varrette (University of Luxembourg)
Interlude: Software Management in HPC systems

Software/Modules Management

Based on Environment Modules / LMod
   - convenient way to dynamically change the users environment $PATH$
   - permits to easily load software through module command

Currently on UL HPC:
   - > 163 software packages, in multiple versions, within 18 categ.
   - reworked software set for iris cluster and now deployed everywhere
     - RESIF v2.0, allowing [real] semantic versioning of released builds
   - hierarchical organization   Ex: toolchain/\{foss,intel\}

$>\text{module\ avail}\quad\text{# List available modules}$

$>\text{module\ load\ <category>/<software>[/<version>]}$
**Key module variable:** $MODULEPATH / where to look for modules

→ altered with module use <path>. **Ex:**

```bash
export EASYBUILD_PREFIX=$HOME/.local/easybuild
export LOCAL_MODULES=$EASYBUILD_PREFIX/modules/all
module use $LOCAL_MODULES
```
Interlude: Software Management in HPC systems

Software/Modules Management

- Key module variable: `$MODULEPATH` / where to look for modules
  - altered with `module use <path>`. Ex:

```
export EASYBUILD_PREFIX=$HOME/.local/easybuild
export LOCAL_MODULES=$EASYBUILD_PREFIX/modules/all
module use $LOCAL_MODULES
```

Main modules commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>module avail</code></td>
<td>Lists all the modules which are available to be loaded</td>
</tr>
<tr>
<td><code>module spider &lt;pattern&gt;</code></td>
<td>Search for among available modules (Lmod only)</td>
</tr>
<tr>
<td><code>module load &lt;mod1&gt; [mod2...]</code></td>
<td>Load a module</td>
</tr>
<tr>
<td><code>module unload &lt;module&gt;</code></td>
<td>Unload a module</td>
</tr>
<tr>
<td><code>module list</code></td>
<td>List loaded modules</td>
</tr>
<tr>
<td><code>module purge</code></td>
<td>Unload all modules (purge)</td>
</tr>
<tr>
<td><code>module display &lt;module&gt;</code></td>
<td>Display what a module does</td>
</tr>
<tr>
<td><code>module use &lt;path&gt;</code></td>
<td>Prepend the directory to the MODULEPATH environment variable</td>
</tr>
<tr>
<td><code>module unuse &lt;path&gt;</code></td>
<td>Remove the directory from the MODULEPATH environment variable</td>
</tr>
</tbody>
</table>
Interlude: Software Management in HPC systems

Software/Modules Management

**Easybuild**: open-source framework to (automatically) build scientific software

**Why?**: "Could you please install this software on the cluster?"

- Scientific software is often **difficult** to build
  - non-standard build tools / incomplete build procedures
  - hardcoded parameters and/or poor/outdated documentation

- EasyBuild helps to facilitate this task
  - **consistent** software build and installation framework
  - includes testing step that helps validate builds
  - **automatically** generates LMod modulefiles

```bash
$> module use $LOCAL_MODULES
$> module load tools/EasyBuild
$> eb -S HPL  # Search for recipes for HPL software
$> eb HPL-2.2-intel-2017a.eb # Install HPL 2.2 w. Intel toolchain
```
Interlude: Software Management in HPC systems

Hands-on 1: Modules & Easybuild

Your Turn!

Hands-on 1


- **Discover** Environment Modules and Lmod
- **Installation of** EasyBuild
- **Local** vs. **Global** Usage
  - local installation of **zlib**
  - global installation of **snappy** and **protobuf**, needed later

Part 1

Part 2 (a)

Part 2 (b)
Interlude: Software Management in HPC systems

Hands-on 2: Building Hadoop

- We will need to install the Hadoop MapReduce by Cloudera using EasyBuild.
  - this build is quite long (~30 minutes on 4 cores)
  - **Obj:** make it build while the keynote continues ;)

Hands-on 2


- **Pre-requisites**
  - Installing **Java 1.7.0** (7u80) and **1.8.0** (8u152)
  - Installing **Maven 3.5.2**
  - Installing **Hadoop 2.6.0-cdh5.12.0**

Sebastien Varrette  (University of Luxembourg)
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Data Intensive Computing

- Data volumes increasing massively
  - Clusters, storage capacity increasing massively
- Disk speeds are not keeping pace.
- Seek speeds even worse than read/write
Data volumes increasing massively

- Clusters, storage capacity increasing massively
- Disk speeds are not keeping pace.
- Seek speeds even worse than read/write
Speed Expectation on Data Transfer

- How long to transfer **1 TB** of data across various speed networks?

<table>
<thead>
<tr>
<th>Network</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Mbps</td>
<td>300 hrs (12.5 days)</td>
</tr>
<tr>
<td>100 Mbps</td>
<td>30 hrs</td>
</tr>
<tr>
<td>1 Gbps</td>
<td>3 hrs</td>
</tr>
<tr>
<td>10 Gbps</td>
<td>20 minutes</td>
</tr>
</tbody>
</table>

- (Again) small I/Os really **kill** performances
  - Ex: transferring 80 TB for the backup of ecosystem_biology
  - same rack, 10Gb/s. 4 weeks → 63TB transfer...
## Speed Expectation on Data Transfer

<table>
<thead>
<tr>
<th>Data set size</th>
<th>1 Minute</th>
<th>5 Minutes</th>
<th>20 Minutes</th>
<th>1 Hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>10PB</td>
<td>166.67 TB/sec</td>
<td>33.33 TB/sec</td>
<td>8.33 TB/sec</td>
<td>2.78 TB/sec</td>
</tr>
<tr>
<td>1PB</td>
<td>16.67 TB/sec</td>
<td>3.33 TB/sec</td>
<td>833.33 GB/sec</td>
<td>277.78 GB/sec</td>
</tr>
<tr>
<td>100TB</td>
<td>1.67 TB/sec</td>
<td>333.33 GB/sec</td>
<td>83.33 GB/sec</td>
<td>27.78 GB/sec</td>
</tr>
<tr>
<td>10TB</td>
<td>166.67 GB/sec</td>
<td>33.33 GB/sec</td>
<td>8.33 GB/sec</td>
<td>2.78 GB/sec</td>
</tr>
<tr>
<td>1TB</td>
<td>16.67 GB/sec</td>
<td>3.33 GB/sec</td>
<td>833.33 MB/sec</td>
<td>277.78 MB/sec</td>
</tr>
<tr>
<td>100GB</td>
<td>1.67 GB/sec</td>
<td>333.33 MB/sec</td>
<td>83.33 MB/sec</td>
<td>27.78 MB/sec</td>
</tr>
<tr>
<td>10GB</td>
<td>166.67 MB/sec</td>
<td>33.33 MB/sec</td>
<td>8.33 MB/sec</td>
<td>2.78 MB/sec</td>
</tr>
<tr>
<td>1GB</td>
<td>16.67 MB/sec</td>
<td>3.33 MB/sec</td>
<td>0.83 MB/sec</td>
<td>0.28 MB/sec</td>
</tr>
<tr>
<td>100MB</td>
<td>1.67 MB/sec</td>
<td>0.33 MB/sec</td>
<td>0.08 MB/sec</td>
<td>0.03 MB/sec</td>
</tr>
</tbody>
</table>

http://fasterdata.es.net/
## Speed Expectation on Data Transfer

<table>
<thead>
<tr>
<th>Data set size</th>
<th>1XB</th>
<th>100PB</th>
<th>10PB</th>
<th>1PB</th>
<th>10TB</th>
<th>100TB</th>
<th>1TB</th>
<th>100GB</th>
<th>10GB</th>
<th>8 Hours</th>
<th>24 Hours</th>
<th>7 Days</th>
<th>30 Days</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>34.72 TB/sec</td>
<td>11.57 TB/sec</td>
<td>1.65 TB/sec</td>
<td>385.80 GB/sec</td>
<td>3.47 TB/sec</td>
<td>11.57 TB/sec</td>
<td>1.65 TB/sec</td>
<td>385.80 MB/sec</td>
<td>385.80 MB/sec</td>
<td>3.47 GB/sec</td>
<td>11.57 GB/sec</td>
<td>1.65 GB/sec</td>
<td>385.80 MB/sec</td>
</tr>
<tr>
<td>100PB</td>
<td>3.47 TB/sec</td>
<td>1.16 TB/sec</td>
<td>165.34 GB/sec</td>
<td>38.58 GB/sec</td>
<td>3.47 GB/sec</td>
<td>1.16 GB/sec</td>
<td>165.34 MB/sec</td>
<td>38.58 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.04 MB/sec</td>
<td></td>
</tr>
<tr>
<td>10PB</td>
<td>347.22 GB/sec</td>
<td>115.74 GB/sec</td>
<td>16.53 GB/sec</td>
<td>3.86 GB/sec</td>
<td>347.22 GB/sec</td>
<td>115.74 GB/sec</td>
<td>16.53 MB/sec</td>
<td>3.86 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.02 MB/sec</td>
<td></td>
</tr>
<tr>
<td>1PB</td>
<td>347.22 GB/sec</td>
<td>115.74 GB/sec</td>
<td>16.53 GB/sec</td>
<td>3.86 GB/sec</td>
<td>347.22 GB/sec</td>
<td>115.74 GB/sec</td>
<td>16.53 MB/sec</td>
<td>3.86 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.02 MB/sec</td>
<td></td>
</tr>
<tr>
<td>10TB</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.04 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.04 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.04 MB/sec</td>
<td></td>
</tr>
<tr>
<td>1TB</td>
<td>347.22 MB/sec</td>
<td>115.74 MB/sec</td>
<td>16.53 MB/sec</td>
<td>3.86 MB/sec</td>
<td>347.22 MB/sec</td>
<td>115.74 MB/sec</td>
<td>16.53 MB/sec</td>
<td>3.86 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.02 MB/sec</td>
<td></td>
</tr>
<tr>
<td>100GB</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.04 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.04 MB/sec</td>
<td>3.47 MB/sec</td>
<td>1.16 MB/sec</td>
<td>0.17 MB/sec</td>
<td>0.00 MB/sec</td>
<td></td>
</tr>
<tr>
<td>10GB</td>
<td>0.35 MB/sec</td>
<td>0.12 MB/sec</td>
<td>0.02 MB/sec</td>
<td>0.00 MB/sec</td>
<td>0.35 MB/sec</td>
<td>0.12 MB/sec</td>
<td>0.02 MB/sec</td>
<td>0.00 MB/sec</td>
<td>0.35 MB/sec</td>
<td>0.12 MB/sec</td>
<td>0.02 MB/sec</td>
<td>0.00 MB/sec</td>
<td></td>
</tr>
</tbody>
</table>

http://fasterdata.es.net/
Storage Performances: GPFS

![Graph showing I/O bandwidth vs. Number of nodes for write and read operations with file sizes of 20G.](image-url)
Storage Performances: Lustre

- Write, filesize 20G
- Read, filesize 20G

Number of nodes vs. I/O bandwidth (MiB/s)
Storage Performances

- Based on IOR or IOZone, reference I/O benchmarks

  → tests performed in 2013

![Graph showing I/O bandwidth (MiB/s) vs. Number of threads for different storage types: SHM / Bigmem, Lustre / Gaia, NFS / Gaia, SSD / Gaia, Hard Disk / Chaos.](image-url)
Based on IOR or IOZone, reference I/O benchmarks  

*tests performed in 2013*
### Understanding Your Storage Options

Where can I store and manipulate my data?

- **Shared storage**
  - NFS - **not scalable** ~ 1.5 GB/s (R) \(\mathcal{O}(100 \ TB)\)
  - GPFS - **scalable** ~ 10 GB/s (R) \(\mathcal{O}(1 \ PB)\)
  - Lustre - **scalable** ~ 5 GB/s (R) \(\mathcal{O}(0.5 \ PB)\)

- **Local storage**
  - local file system (/tmp) \(\mathcal{O}(200 \ GB)\)
    - over HDD ~ 100 MB/s, over SDD ~ 400 MB/s
  - RAM (/dev/shm) ~ 30 GB/s (R) \(\mathcal{O}(20 \ GB)\)

- **Distributed storage**
  - HDFS, Ceph, GlusterFS - **scalable** ~ 1 GB/s

⇒ **In all cases:** small I/Os really **kill** storage performances
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Data Transfer in Practice

$>$ wget [-O <output>] <url>  # download file from <url>

$>$ curl [-o <output>] <url>  # download file from <url>

- Transfer **from** FTP/HTTP[S]  
  - can also serve to send HTTP POST requests  
  - support HTTP cookies (useful for JDK download)

wget or (better) curl
Data Transfer in Practice

$> \text{scp } [-P <\text{port}>] \ <\text{src}> \ <\text{user}@<\text{host}>:<\text{path}>$

$> \text{rsync } -avzu [-e 'ssh -p <\text{port}>'] \ <\text{src}> \ <\text{user}@<\text{host}>:<\text{path}>

- [Secure] Transfer \textbf{from/to} two remote machines over SSH
  \rightarrow \text{scp} \text{ or (better) } \text{rsync} \ (\text{transfer } \textbf{only} \text{ what is required})
- Assumes you have understood and configured appropriately SSH!
SSH: Secure Shell

- Ensure **secure** connection to remote (UL) server
  - establish **encrypted** tunnel using **asymmetric keys**
    - **Public** id_rsa.pub vs. **Private** id_rsa (**without** .pub)
    - typically on a non-standard port (**Ex**: 8022) **limits kiddie script**
    - Basic rule: 1 machine = 1 key pair
  - the private key is **SECRET**: **never** send it to anybody
    - Can be protected with a passphrase
**SSH: Secure Shell**

- **Ensure secure connection to remote (UL) server**
  - Establish **encrypted** tunnel using **asymmetric keys**
    - **Public** id_rsa.pub vs. **Private** id_rsa (without .pub)
    - Typically on a non-standard port (**Ex:** 8022) limits kiddie script
    - Basic rule: 1 machine = 1 key pair
  - The private key is **SECRET**: *never* send it to anybody
    - Can be protected with a passphrase

- **SSH** is used as a secure backbone channel for **many** tools
  - Remote shell *i.e.* remote command line
  - File transfer: **rsync**, **scp**, **sftp**
  - Versionning synchronization (**svn**, **git**), **github**, **gitlab** etc.
SSH: Secure Shell

- Ensure **secure** connection to remote (UL) server
  - establish **encrypted** tunnel using **asymmetric keys**
    - **Public** id_rsa.pub vs. **Private** id_rsa (without .pub)
    - typically on a non-standard port (Ex: 8022) limits kiddie script
    - Basic rule: 1 machine = 1 key pair
  - the private key is **SECRET**: never send it to anybody
    - Can be protected with a passphrase

- SSH is used as a secure backbone channel for **many** tools
  - Remote shell *i.e.* remote command line
  - File transfer: rsync, scp, sftp
  - versionning synchronization (svn, git), github, gitlab etc.

- Authentication:
  - **password** (disable if possible)
  - **(better) public key authentication**
SSH: Public Key Authentication

- Local machine owns local private key.
- Logs known servers.
- Key files:
  - `id_rsa`
  - `id_rsa.pub`
  - `known_hosts`

Restrictions:
- `PermitRootLogin no`
- `PasswordAuthentication no`
- `ChallengeResponseAuthentication no`
- `RSAAuthentication yes`
- `PubkeyAuthentication yes`
SSH: Public Key Authentication

Client
Local Machine

- local homedir
- ~/.ssh/
- id_rsa
- id_rsa.pub
- known_hosts

owns local private key

logs known servers

Server
Remote Machine

- remote homedir
- ~/.ssh/
- authorized_keys

knows granted (public) key
**SSH: Public Key Authentication**

**Client (Local Machine)**
- **~/.ssh/**
  - `id_rsa`
  - `id_rsa.pub`
  - `known_hosts`
  - owns local private key
  - logs known servers

**Server (Remote Machine)**
- **~/.ssh/**
  - `authorized_keys`
  - knows granted (public) key
- **/etc/ssh/**
  - `sshd_config`
  - `ssh_host_rsa_key`
  - `ssh_host_rsa_key.pub`

Restrict to public key authentication:
- `PermitRootLogin no`
- `PasswordAuthentication no`
- `ChallengeResponseAuthentication no`
- `RSAAuthentication yes`
- `PubkeyAuthentication yes`
SSH: Public Key Authentication

Client Local Machine

- local homedir
- ~/.ssh/
- id_rsa
- id_rsa.pub

owns local private key

Server Remote Machine

- remote homedir
- ~/.ssh/
- authorized_keys

knows granted (public) key

Restrict to public key authentication:

- PermitRootLogin no
- PasswordAuthentication no
- ChallengeResponseAuthentication no
- RSAAuthentication yes
- PubkeyAuthentication yes
**SSH: Public Key Authentication**

1. Initiate connection
2. Create random challenge, "encrypt" using public key
3. Solve challenge using private key, return response
4. Allow connection if response == challenge

- **Restrict** to public key authentication: `/etc/ssh/sshd_config`:

  ```
  PermitRootLogin no
  # Disable Passwords
  PasswordAuthentication no
  ChallengeResponseAuthentication no
  # Enable Public key auth.
  RSAAuthentication yes
  PubkeyAuthentication yes
  ```
Hands-on 3: Data transfer over SSH

Before doing Big Data, learn how to transfer data between 2 hosts

→ do it securely over SSH

# Quickly generate a 10GB file

$> dd if=/dev/zero of=/tmp/bigfile.txt bs=100M count=100

# Now try to transfer it between the 2 Vagrant boxes ;)

Hands-on 3


- Generate SSH Key Pair and authorize the public part  Step 1
- Data transfer over SSH with scp  Step 2.a
- Data transfer over SSH with rsync  Step 2.b
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Before doing Big Data, manage and version correctly normal data

What kinds of systems are available?

- **Good**: NAS, Cloud, Dropbox, Google Drive, Figshare...
- **Better** - **Version Control systems** (VCS)
  - SVN, Git, and Mercurial
- **Best** - **Version Control Systems** on the Public/Private Cloud
  - GitHub, Bitbucket, Gitlab
Before doing **Big** Data, manage and version correctly **normal** data.

### What kinds of systems are available?

- **Good**: NAS, Cloud, Dropbox, Google Drive, Figshare...
- **Better** - Version Control systems (VCS)
  - SVN, Git and Mercurial
- **Best** - Version Control Systems on the Public/Private Cloud
  - GitHub, Bitbucket, Gitlab

**Which one?**
- Depends on the level of privacy you expect
  - ✓ ... but you probably already know these tools 😊
  - Few handle GB files...
Centralized VCS - CVS, SVN
Centralized VCS - CVS, SVN
Everybody has the full history of commits
Tracking changes (most VCS)

Checkins over Time

- C1
- file A
- file B
- file C
Tracking changes (most VCS)

Checkins over Time

- C1
- C2
- file A → Δ1
- file B
- file C → Δ1
Tracking changes (most VCS)

Checkins over Time

C1 → Δ1
file A

C2
Δ1
file B

C3 → Δ2
file C
Tracking changes (most VCS)

Checkins over Time

C1 → ∆1

file A

C2 → ∆1

file B

C3 → ∆2

file C

C4 → ∆1

Sebastien Varrette (University of Luxembourg)
Tracking changes (most VCS)

Checkins over Time

- C1 → Δ1 → C2 → Δ2 → C3
- file A
- file B
- file C

- C4 → Δ1 → C5 → Δ2 → Δ3

Sebastien Varrette (University of Luxembourg)
Tracking changes (most VCS)
Tracking changes (Git)

Checkins over Time

- C1
- file A
- Δ1
- Δ2
- C2
- file B
- Δ1
- Δ2
- C3
- file C
- Δ1
- Δ2
- C4
- Δ2
- C5

delta storage

snapshot (DAG) storage
Tracking changes (Git)

- **Delta storage**
  - file A → \(\Delta_1\) → \(\Delta_2\)
  - file B → \(\Delta_1\) → \(\Delta_2\)
  - file C → \(\Delta_1\) → \(\Delta_2\) → \(\Delta_3\)

- **Snapshot (DAG) storage**
  - C1
  - A
  - B
  - C
Tracking changes (Git)
Tracking changes (Git)

Checkins over Time

delta storage

file A \( \Delta 1 \) file B \( \Delta 1 \) file C \( \Delta 1 \)

\( \Delta 2 \) \( \Delta 1 \) \( \Delta 2 \)

\( \Delta 3 \)

snapshot (DAG) storage

A \( A1 \) B \( B \) C \( C1 \)
Tracking changes (Git)

Checkins over Time

delta storage

snapshot (DAG) storage
Tracking changes (Git)

- Checkins over Time
  - C1 → C2 → C3 → C4 → C5
  - file A → Δ1 → Δ2
  - file B → Δ1 → Δ2
  - file C → Δ1 → Δ2

- delta storage

- snapshot (DAG) storage

Sebastien Varrette (University of Luxembourg)
Tracking changes (Git)

Checkins over Time

delta storage

file A → Δ1 → Δ2
file B → Δ1 → Δ2
file C → Δ1 → Δ2

snapshot (DAG) storage

A → A1 → A1
B → B → B
C → C1 → C2

Checkins over Time
Tracking changes (Git)

- **Checkins over Time**
- **delta storage**
- **snapshot (DAG) storage**

- File A → Δ1 → Δ2
- File B → Δ1 → Δ2
- File C → Δ1 → Δ2
- File C → Δ2 → Δ3

- Checkins over Time diagram with changes in files and storage structures.
[Big] Data Management in HPC Environment: Overview and Challenges

Tracking changes (Git)

Checkins over Time

delta
storage

C1 → Δ1 → Δ2
file A

Δ1
Δ2

file B

Δ1
Δ2

file C

Checkins over Time

snapshot
(DAG)
storage

C1 → A → A1 → B → B1 → C1

A1
A2

B
B1
B2

C
C1
C2
C3

C1
C2
C3
C4
C5

62 / 133

Sebastien Varrette (University of Luxembourg)
Tracking changes (Git)

Checkins over Time

delta storage

file A

Δ1

file B

Δ1

file C

Δ1

Checkins over Time

snapshot (DAG) storage

A

A1

B

B

C

C1

A

A1

B

B

C

C1

B

B1

C

C2

C

C2

B

B2

C

C3
**VCS Taxonomy**

- **Delta Storage**
  - Local
  - Centralized
  - Distributed
  - RCS
  - CVS
  - Mercurial

- **Snapshot (DAG) Storage**
  - Local
  - Centralized
  - Distributed
  - Git
  - Bazaar
  - Rsync
  - Duplication
  - Time Machine
  - Backup Ninja
  - Duplicity

- **Mac OS File Versions**
Git at the heart of BD

http://git-scm.org
(Reference) web-based Git repository hosting service

Set up Git

Create Repository

Fork repository

Work together

Sebastien Varrette (University of Luxembourg)
So what makes Git so useful?

(almost) Everything is local

- everything is fast
- every clone is a backup
- you work *mainly offline*

Ultra Fast, Efficient & Robust

- Snapshots, not patches (deltas)
- **Cheap branching and merging**
  - Strong support for thousands of parallel branches
- Cryptographic integrity everywhere
Other Git features

- **Git does not delete**
  - Immutable objects, Git generally only adds data
  - If you mess up, you can usually recover your stuff
    - Recovery can be tricky though
Other Git features

- **Git does not delete**
  - Immutable objects, Git generally only adds data
  - If you mess up, you can usually recover your stuff
    ✓ Recovery can be tricky though

### Git Tools / Extension

- cf. **Git submodules** or **subtrees**
- **Introducing git-flow**
  - workflow with a strict branching model
  - offers the git commands to follow the workflow

```bash
$> git flow init
$> git flow feature { start, publish, finish } <name>
$> git flow release { start, publish, finish } <version>
```
Git in practice

**Basic Workflow**

- **Pull** latest changes
- **Edit** files  
  - `vim` / `emacs` / `subl` ...
- **Stage** the changes
- **Review** your changes
- **Commit** the changes

```
git pull
vim / emacs / subl ...
git add

```

```
git status

```

```
git commit

```
Git in practice

**Basic Workflow**

- **Pull** latest changes
- **Edit** files
- **Stage** the changes
- **Review** your changes
- **Commit** the changes

**For cheaters: A Basicerer Workflow**

- **Pull** latest changes
- **Edit** files
- **Stage & commit all the changes**

```bash
# For cheaters: A Basicerer Workflow

Pull latest changes
vim / emacs / subl ...

Edit files
git add

global view

Stage & commit all the changes
git status

Commit the changes
git commit

```
Git Summary

**Advices**: Commit early, commit often!

- commits = save points
  - use descriptive commit messages
- Do not get out of sync with your collaborators
- Commit the sources, not the derived files

**Not covered here (by lack of time)**

- does not mean you should not dig into it!
- **Resources**:
  - https://git-scm.com/
  - tutorial: Reproducible Research at the Cloud Era
Summary

1 Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2 Interlude: Software Management in HPC systems

3 [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4 Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5 Deep Learning Analytics with Tensorflow
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
What is a Distributed File System?

- **Straightforward idea**: separate logical from physical storage.
  - Not all files reside on a single physical disk,
  - or the same physical server,
  - or the same physical rack,
  - or the same geographical location,…

- **Distributed file system (DFS)**:
  - virtual file system that enables clients to access files
    - … as if they were stored locally.
What is a Distributed File System?

- Straightforward idea: separate logical from physical storage.
  - Not all files reside on a single physical disk,
  - or the same physical server,
  - or the same physical rack,
  - or the same geographical location,…

- Distributed file system (DFS):
  - virtual file system that enables clients to access files
    ✓ … as if they were stored locally.

- Major DFS distributions:
  - NFS: originally developed by Sun Microsystems, started in 1984
  - AFS/CODA: originally prototypes at Carnegie Mellon University
  - GFS: Google paper published in 2003, not available outside Google
  - HDFS: designed after GFS, part of Apache Hadoop since 2006
Distributed File System Architecture?

### Master-Slave Pattern

- Single (or few) **master** nodes maintain state info. about clients
- All clients R&W requests go through the global master node.
- **Ex:** GFS, HDFS
Distributed File System Architecture?

**Master-Slave Pattern**

- Single (or few) **master** nodes maintain state info. about clients.
- All clients R&W requests go through the global master node.
- **Ex:** GFS, HDFS

**Peer-to-Peer Pattern**

- No global state information.
- Each node may both serve and process data.
Radically different architecture compared to NFS, AFS and CODA.

- specifically tailored towards **large-scale** and **long-running analytical processing tasks**
- over thousands of storage nodes.

**Basic assumption:**

- client nodes (aka. *chunk servers*) may fail any time!
- Bugs or hardware failures.
- Special tools for monitoring, periodic checks.
- Large files (multiple GBs or even TBs) are split into 64 MB *chunks*.
- Data modifications are mostly append operations to files.
- Even the master node may fail any time!
  - Additional *shadow master* fallback with read-only data access.

**Two types of reads:** Large sequential reads & small random reads
GFS Consistency Model

- **Atomic File Namespace Mutations**
  - File creations/deletions centrally controlled by the master node.
  - Clients typically create and write entire file,
    - then add the file name to the file namespace stored at the master.

- **Atomic Data Mutations**
  - only 1 atomic modification of 1 replica (!) at a time is guaranteed.

- **Stateful Master**
  - Master sends regular **heartbeat** messages to the chunk servers
  - Master keeps chunk locations of all files (+ replicas) in memory.
  - locations not stored persistently...
    - but polled from the clients at startup.

- **Session Semantics**
  - Weak consistency model for file replicas and client caches only.
  - Multiple clients may read and/or write the same file concurrently.
  - The client that last writes to a file **wins**.
Fault Tolerance & Fault Detection

- **Fast Recovery**
  - master & chunk servers can restore their states and (re-)start in s.
  - ✓ regardless of previous termination conditions.

- **Master Replication**
  - shadow master provides RO access when primary master is down.
  - ✓ Switches back to read/write mode when primary master is back.
  - Master node does not keep a persistent state info. of its clients,
  - ✓ rather polls clients for their states when started.

- **Chunk Replication & Integrity Checks**
  - chunk divided into 64 KB blocks, each with its own 32-bit checksum
  - ✓ verified at read and write times.
  - ➡ Higher replication factors for more intensively requested chunks (hotspots) can be configured.
Map-Reduce

- Breaks the processing into two main phases:
  1. the **map** phase
  2. the **reduce** phase.

- Each phase has key-value pairs as input and output,
  - the types of which may be chosen by the programmer.
  - the programmer also specifies the **map** and **reduce** functions.
**Hadoop**

- Initially started as a student project at Yahoo! labs in 2006
  - Open-source Java implem. of GFS and MapReduce frameworks
- Switched to Apache in 2009. Now consists of three main modules:
  1. **HDFS**: Hadoop distributed file system
  2. **YARN**: Hadoop job scheduling and resource allocation
  3. **MapReduce**: Hadoop adaptation of the MapReduce principle
Hadoop

- Initially started as a student project at Yahoo! labs in 2006
  - Open-source Java implem. of GFS and MapReduce frameworks
- Switched to Apache in 2009. Now consists of three main modules:
  1. **HDFS**: Hadoop distributed file system
  2. **YARN**: Hadoop job scheduling and resource allocation
  3. **MapReduce**: Hadoop adaptation of the MapReduce principle

- Basis for many other open-source Apache toolkits:
  - **PIG/PigLatin**: file-oriented data storage & script-based query language
  - **HIVE**: distributed SQL-style data warehouse
  - **HBase**: distributed key-value store
  - **Cassandra**: fault-tolerant distributed database, etc.

- HDFS still mostly follows the original GFS architecture.
HDD streaming speed $\sim 50\text{MB/s}$
- $3\text{TB} = 17.5 \text{hrs}$
- $1\text{PB} = 8 \text{months}$

Scale-out (weak scaling)
- FS distributes data on ingest

Seeking too slow
- $\sim 10\text{ms for a seek}$
- Enough time to read half a megabyte

Batch processing
- Go through entire data set in one (or small number) of passes
Combining Results

- Each node preprocesses its local data
  - Shuffles its data to a small number of other nodes
- Final processing, output is done there
Fault Tolerance

- Data also replicated upon ingest
- Runtime watches for dead tasks, restarts them on live nodes
- Re-replicates
Hadoop: What is it Good At?

- **Classic Hadoop 1.x is all about batch processing** of massive amounts of data;
  - Not much point below ~1TB
- **Map-Reduce** is relatively loosely coupled;
  - one **shuffle** phase.
- **Very strong weak scaling in this model**
  - more data, more nodes.
- **Batch:**
  - process all data in one go
    - w/classic Map Reduce
  - Current Hadoop has many other capabilities besides batch - **more later**
Hadoop: What is it Good At?

- **Compare with databases**
  - → very good at working on small subsets of large databases
    - ✓ DBs: very interactive for many tasks
    - ✓ . . . yet have been difficult to scale

- **Compare with HPC (MPI)**
  - → Also typically batch
  - → Can (and does) go up to enormous scales

- **Works extremely well for very tightly coupled problems:**
  - → zillions of iterations/timesteps/exchanges.
We HPC users might be tempted to an unseemly smugness

They solved the problem of disk-limited, loosely-coupled, data analysis by throwing more disks at it and weak scaling?

Ooooooooh

We would be wrong.

A single novice developer can write:

- real, scalable,
- 1000+ node data-processing tasks in Hadoop-family tools in an afternoon.

In MPI... less likely...
Data Distribution: Disk

- Hadoop & al. arch. handle the hardest part of parallelism for you
  - aka data distribution.

- **On disk:**
  - HDFS distributes, replicates data as it comes in
  - Keeps track of computations local to data
On network: Map Reduce (eg) works in terms of key-value pairs.

- Preprocessing (map) phase ingests data, emits \((k, v)\) pairs
- Shuffle phase assigns reducers,
  - gets all pairs with same key onto that reducer.
- Programmer does not have to design communication patterns
Big Data Analytics with Hadoop & Spark

Makes the problem easier

- Hardest parts of parallel programming with HPC tools
  - Decomposing the problem, and,
  - Getting the intermediate data where it needs to go,

- Hadoop does that for you
  - automatically
  - for a wide range of problems.
Built a reusable substrate

- HDFS and the MapReduce layer were very well architected.
  - Enables many higher-level tools
  - Data analysis, machine learning, NoSQL DBs, ...
- Extremely productive environment
  - And Hadoop 2.x (YARN) is now much much more than just MapReduce
Hadoop and HPC

- Not either-or anyway
  - Use HPC to generate big / many simulations,
  - Use Hadoop to analyze results
    - Ex: Use Hadoop to preprocess huge input data sets (ETL),
    - ... and HPC to do the tightly coupled computation afterwards.

- In all cases: Everything is Converging
HDFS is a distributed parallel filesystem

- Not a general purpose file system
  - does not implement posix
  - cannot just mount it and view files

Access via hdfs fs commands or programatic APIs

Security slowly improving

$> hdfs fs -[cmd]
The Hadoop Filesystem

- **Required** to be:
  - able to deal with large files, large amounts of data
  - scalable & reliable in the presence of failures
  - fast at reading contiguous streams of data
  - only need to write to new files or append to files
  - require only commodity hardware

- **As a result**:
  - Replication
  - Supports mainly high bandwidth, **not** especially low latency
  - No caching
    - what is the point if primarily for streaming reads?
    - Poor support for seeking around files
    - Poor support for zillions of files
  - Have to use separate API to see filesystem
  - Modelled after Google File System (2004 Map Reduce paper)
Hadoop vs HPC

- **HDFS is a block-based** FS
  - A file is broken into blocks,
  - these blocks are distributed across nodes

- **Blocks are large**;
  - 64MB is default,
  - many installations use 128MB or larger

- **Large block size**
  - time to stream a block much larger than time disk time to access the block.

# Lists all blocks in all files:

```
$ hdfs fsck / -files -blocks
```

Sebastien Varrette (University of Luxembourg)
Two types of nodes in the filesystem:

1. **Namenode**
   - stores all metadata / block locations in memory
   - Metadata updates stored to persistent journal

2. **Datanodes**
   - store/retrieve blocks for client/namenode

- Newer versions of Hadoop: federation
  - ≠ namenodes for /user, /data...
  - High Availability namenode pairs
Writing a file

Writing a file multiple stage process:

- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete
**Writing a file**

- **Client**: Write newdata.dat

2. **get nodes**

**Namenode**

- `/user/lpurs/oldhze`
- `newdata.dat`

- **Writing a file** multiple stage process:
  - Create file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get ack back
  - Complete
Writing a file

- Writing a file multiple stage process:
  - Create file
  - Get nodes for blocks
  - Start writing
  - Data nodes coordinate replication
  - Get ack back
  - Complete
Writing a file multiple stage process:

- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back
- Complete
Writing a file multiple stage process:

- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back (while writing)
- Complete
Writing a file multiple stage process:

- Create file
- Get nodes for blocks
- Start writing
- Data nodes coordinate replication
- Get ack back *(while writing)*
- Complete
Where to Replicate?

- **Tradeoff** to choosing replication locations
  - **Close**: faster updates, less network bandwidth
  - **Further**: better failure tolerance

- **Default strategy**:
  1. copy on different location on same node
  2. second on different *rack* (switch),
  3. third on same rack location, different node.

- **Strategy configurable**.
  - Need to configure Hadoop file system to know location of nodes
Reading a file

1. Open

- Open call
- Get block locations
- Read from a replica

Client:
Read lines 1...1000 from bigdata.dat

Namenode

Datanode 1
Datanode 2
Datanode 3
Reading a file

1. Client:
   Read lines 1...1000 from bigdata.dat

2. Get block locations

- Open call
- Get block locations
- Read from a replica
Reading a file

1. Open call
2. Get block locations
3. Read from a replica
Configuring HDFS

- Need to tell HDFS how to set up filesystem
  - `data.dir`, `name.dir`
    - ✓ where on local system (eg, local disk) to write data
  - parameters like replication
    - ✓ how many copies to make
  - default name - default file system to use
  - Can specify multiple FSs
<!-- $HADOOP_PREFIX/etc/hadoop/core-site.xml -->

<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://<server>:9000</value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>/home/username/hdfs/data</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>/home/username/hdfs/name</value>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
  </property>
</configuration>
In Practice, in single mode

- Only one node to be used, the VM
- **default server**: localhost
- Since only one node:
  - ✓ need to specify replication factor of 1, or will always fail

```xml
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>

[...]

<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```
You will need to make sure that environment variables are set

- path to Java, path to Hadoop...
- Easybuild does *most* of the job for you

You will need passwordless SSH access across all nodes

You can then start processes on various FS nodes
Configuring HDFS

- You will need to make sure that environment variables are set
  → path to Java, path to Hadoop...
  → Easybuild does most of the job for you
- You will need passwordless SSH access across all nodes
- You can then start processes on various FS nodes

Once configuration files are set up,
  → you can format the namenode like so
  → you can start up just the file systems

```bash
$> hdfs namenode -format
$> start-dfs.sh
```
Using HDFS

Once the file system is up and running,

→ ... you can copy files back and forth

Default directory is /user/${username}

→ Nothing like a cd
In general, the data files you send to HDFS will be \textbf{large} or else why bother with Hadoop.

Do not want to be constantly copying back and forth → \textit{view, append in place}

Several APIs to accessing the HDFS → Java, C++, Python

Here, we use one to get a file status, and read some data from it at some given offset
Map processes **one element at a time**

- emits results as (key, value) pairs.

All results with **same key are gathered to the same reducers**

- Reducers process list of values
- emit results as (key, value) pairs
You have uploaded a slide from a presentation on Big Data Analytics with Hadoop & Spark. The slide focuses on the concept of Map, a key component in processing big data.

- **Map**
  - All coupling done during **shuffle** phase
    - Embarrassingly parallel task
    - All map
  - Take input, map it to output, done.
  - **Famous case**
    - NYT using Hadoop to convert 11 million image files to PDFs
      - Almost pure serial farm job

The slide also mentions Sebastien Varrette from the University of Luxembourg as the presenter. The text is clear and the diagram effectively illustrates the concept of Map in Hadoop and Spark.
Reducing gives the coupling

In the case of the NYT task:
- not quite embarrassingly parallel:
  - images from multi-page articles
  - Convert a page at a time,
  - gather images with same article id onto node for conversion
**Shuffle**

- **shuffle is part of the Hadoop magic**
  - By default, keys are hashed
  - hash space is partitioned between reducers

- **On reducer:**
  - gathered \((k,v)\) pairs from mappers are sorted by key,
  - then merged together by key
  -Reducer then runs on one \((k,[v])\) tuple at a time

- you can supply your own partitioner
  - Assign **similar** keys to same node
  - Reducer still only sees one \((k, [v])\) tuple at a time.
Example: Wordcount

- Was used as an example in the original MapReduce paper
  - Now basically the **hello world** of map reduce

**Problem description:** Given a set of documents:
- count occurrences of words within these documents

<table>
<thead>
<tr>
<th>file01</th>
<th>file02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>Hello Hadoop</td>
</tr>
<tr>
<td>Bye World</td>
<td>Goodbye Hadoop</td>
</tr>
</tbody>
</table>

```
output/part-00000
```

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello</td>
<td>2</td>
</tr>
<tr>
<td>World</td>
<td>2</td>
</tr>
<tr>
<td>Bye</td>
<td>1</td>
</tr>
<tr>
<td>Hadoop</td>
<td>2</td>
</tr>
<tr>
<td>Goodbye</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Wordcount

- **How would you do this with a huge document?**
  - Each time you see a word:
    - ✓ if it is a new word, add a tick mark beside it,
    - ✓ otherwise add a new word with a tick

- **... But hard to parallelize**
  - pb when updating the list

<table>
<thead>
<tr>
<th>file01</th>
<th>file02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello World</td>
<td>Hello Hadoop</td>
</tr>
<tr>
<td>Bye World</td>
<td>Goodbye Hadoop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>output/part-00000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hello 2</td>
</tr>
<tr>
<td>World 2</td>
</tr>
<tr>
<td>Bye 1</td>
</tr>
<tr>
<td>Hadoop 2</td>
</tr>
<tr>
<td>Goodbye 1</td>
</tr>
</tbody>
</table>
Example: Wordcount

**MapReduce way**

- all hard work done automatically by shuffle

**Map:**

- just emit a 1 for each word you see

**Shuffle:**

- assigns keys (words) to each reducer,
- sends (k,v) pairs to appropriate reducer

**Reducer**

- just has to sum up the ones

---

Sebastien Varrette (University of Luxembourg)
Example: Wordcount

MapReduce way

- all hard work done automatically by shuffle

Map:
- just emit a 1 for each word you see

Shuffle:
- assigns keys (words) to each reducer,
- sends (k,v) pairs to appropriate reducer

Reducer
- just has to sum up the ones
Example: Wordcount

- **MapReduce way**
  - all hard work done automatically by shuffle
- **Map:**
  - just emit a 1 for each word you see
- **Shuffle:**
  - assigns keys (words) to each reducer,
  - sends (k,v) pairs to appropriate reducer
- **Reducer**
  - just has to sum up the ones
Your Turn!

Now you are ready to play with the installed Hadoop

Hands-on 4


- Test the tools/Hadoop modules in Single mode  
  → setup the wordcount example  
  Enable a Cluster Setup  

Step 1

Step 2
Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Hadoop 0.1x

- Original Hadoop was basically HDFS + infra. for MapReduce
  - Very faithful implementation of Google MapReduce paper.
  - Job tracking, orchestration all very tied to M/R model
- Made it difficult to run other sorts of jobs
YARN: Yet Another Resource Negotiator

- Looks a lot more like a cluster scheduler/resource manager
- Allows arbitrary jobs.

Allow for new compute/data tools. Ex: streaming with Spark
Apache Spark

Spark is (yet) a(-nother) distributed, Big Data processing platform. Everything you can do in Hadoop, you can also do in Spark.

In contrast to Hadoop

- Spark computation paradigm is not just MapReduce job
- Key feature - in-memory analyses.
  - multi-stage, in-memory dataflow graph based on Resilient Distributed Datasets (RDDs).
Apache Spark

- Spark is implemented in Scala, running in a Java Virtual Machine.
  - Spark supports different languages for application development:
    - Java, Scala, Python, R, and SQL.
- Originally developed in AMPLab (UC Berkeley) from 2009,
  - donated to the Apache Software Foundation in 2013,
  - top-level project as of 2014.
- **Latest release**: 2.2.1 (Dec. 2017)
Resilient Distributed Dataset (RDD)

- Partitioned collections (lists, maps..) across nodes
- Set of well-defined operations (incl map, reduce) defined on these RDDs.
Fault tolerance works three ways:
- Storing, reconstructing lineage
- Replication (optional)
- Persistence to disk (optional)
RDD Lineage

- Map Reduce implemented FT by outputting everything to disk always.
  - Effective but extremely costly.
  - How to maintain fault tolerance without sacrificing in-memory performance?
  - ✓ for truly large-scale analyses
RDD Lineage

- Map Reduce implemented FT by outputting everything to disk always.
  - Effective but extremely costly.
  - How to maintain fault tolerance without sacrificing in-memory performance?
    ✓ for truly large-scale analyses

**Solution:**

- Record lineage of an RDD (think version control)
- If container, node goes down, reconstruct RDD from scratch
  ✓ Either from beginning,
  ✓ or from (occasional) checkpoints which user has some control over.
- User can suggest caching current state of RDD in memory,
  ✓ or persisting it to disk, or both.
- You can also save RDD to disk, or replicate partitions across nodes for other forms of fault tolerance.
Main Building Blocks

- The **Spark Core API** provides the general execution layer on top of which all other functionality is built upon.
- Four higher-level components (in the _Spark ecosystem_):
  1. **Spark SQL** (formerly **Shark**),
  2. **Streaming**, to build scalable fault-tolerant streaming applications.
  3. **MLlib** for machine learning
  4. **GraphX**, the API for graphs and graph-parallel computation
Hands-on 5: Spark installation

Your Turn!

Hands-on 5


- Use EasyBuild to search for a ReciPY for Apache Spark
- Install it and check the installed software
Big Data Analytics with Hadoop & Spark

Hands-on 6: Spark Usage

Your Turn!

Hands-on 6


- Check a single interactive run
  - PySpark, the Spark Python API
  - Scala Spark Shell
  - R Spark Shell will not be reviewed here.

- Running Spark standalone cluster
  - In particular, illustrated on Pi estimation.

Step 1

Step 1.a.

Step 1.b.

Step 2
Deep Learning Analytics with Tensorflow

Summary

1. Introduction
   Before we start...
   Overview of HPC & BD Trends
   Main HPC and DB Components

2. Interlude: Software Management in HPC systems

3. [Big] Data Management in HPC Environment: Overview and Challenges
   Performance Overview in Data transfer
   Data transfer in practice
   Sharing Data

4. Big Data Analytics with Hadoop & Spark
   Apache Hadoop
   Apache Spark

5. Deep Learning Analytics with Tensorflow
Out-of-scope of this tutorial:
- Machine Learning (ML) / Deep Learning theoretical basis
Machine Learning Algorithms Cheat Sheet

Unsupervised Learning: Clustering
- k-means
- k-modes
- Gaussian Mixture Model
- DBSCAN
- Prefer Probability
- Categorical Variables
- Need to Specify k
- Hierarchical

Unsupervised Learning: Dimension Reduction
- Dimension Reduction
- Principal Component Analysis
- Singular Value Decomposition
- Probabilistic
- Latent Dirichlet Analysis

START
- Have Responses

Supervised Learning: Classification
- Data Is Too Large
- Naïve Bayes
- Linear SVM
- Explainable
- Speed or Accuracy
- Naïve Bayes
- Decision Tree
- Logistic Regression
- Kernel SVM
- Random Forest
- Neural Network
- Gradient Boosting Tree

Supervised Learning: Regression
- Speed or Accuracy
- Decision Tree
- Linear Regression
- Random Forest
- Neural Network
- Gradient Boosting Tree
Deep Learning Analytics with TensorFlow

Machine/Deep-Learning Frameworks

- **Pytorch**
  - Python version of Torch open-sourced by Facebook in 2017.
  - Torch is a computational framework with an API written in Lua that supports machine-learning algorithms.
  - PyTorch offers dynamic computation graphs, which let you process variable-length inputs and outputs.

- **TensorFlow**
  - open source software library from Google for numerical computation using data flow graphs,
  - thus close to the Deep Learning book way of thinking about neural networks.

- **Keras**
  - high-level neural networks API,
  - written in Python and capable of running on top of TensorFlow.

- **Caffe**
  - a well-known and widely used machine-vision library that ported Matlabs implementation of fast convolutional nets to C and C++
  - YoulA21 also have to consider its successor, Caffe 2,
Offer various **Package Design Choices**

- **Model specification:**
  - ✓ Configuration file (Caffe, DistBelief, CNTK) vs. programmatic generation (Torch, Theano, Tensorflow)

- **For programmatic models, choice of high-level language:**
  - ✓ Lua (Torch)
  - ✓ vs. Python (Theano, Tensorflow)
  - ✓ vs others (Go etc.)

**In this talk**

- We chose to work with python because of rich community and library infrastructure.
TensorFlow vs. Theano

- Theano is another deep-learning library with python wrapper
  - was inspiration for TensorFlow

- Theano and TensorFlow are very similar systems.
  - TensorFlow has better support for distributed systems though,
  - development funded by Google, while Theano is an academic project.
Deep Learning Analytics with TensorFlow

What is TensorFlow?

- TensorFlow is a deep learning library recently open-sourced by Google.
  - library for numerical computation using **data flow graphs**.
    - **Nodes** represent mathematical operations,
    - **edges** represent the multidimensional data arrays (**tensors**) communicated between them.

- Flexible architecture allowing to deploy computation anywhere:
  - to one or more CPUs or GPUs in a desktop, server,
  - or mobile device with a single API.

- TensorFlow was originally developed within the Google Brain Team
Deep Learning Analytics with Tensorflow

Hands-on 7: Installing Tensorflow

- **Without further development**
  - You are ready to play with Tensorflow
  - Provided tutorial is self-explicit and make use of Jupyter Notebook

---

Hands-on 7


- Preparation of a Python sand-boxed environment
  - Using *pyenv* and *virtualenv*  
- Tensorflow installation using *pip*  
- Installation of Jupyter *Jupyter Notebook*

---

Sebastien Varrette (University of Luxembourg)
Hands-on 8: Playing with Tensorflow

Your Turn!

Hands-on 8


- Run a very simple MNIST classifier
  - MNIST: computer vision dataset (images of handwritten digits)

- Run a deep MNIST classifier using convolutional layers
  - compare results with best models

Step 1

Step 2
Dr. Sebastien Varrette  
University of Luxembourg, Belval Campus:  
Maison du Nombre, 4th floor  
2, avenue de l'Université  
L-4365 Esch-sur-Alzette  
mail: sebastien.varrette@uni.lu

1 Introduction  
Before we start...  
Overview of HPC & BD Trends  
Main HPC and DB Components

2 Interlude: Software Management in HPC systems

3 [Big] Data Management in HPC Environment: Overview and Challenges  
Performance Overview in Data transfer  
Data transfer in practice  
Sharing Data

4 Big Data Analytics with Hadoop & Spark  
Apache Hadoop  
Apache Spark

5 Deep Learning Analytics with Tensorflow

Thank you for your attention...  
Questions?

http://hpc.uni.lu