Big Data Analytics

3rd NESUS Winter School on Data Science & Heterogeneous Computing

Sébastien Varrette, PhD

Parallel Computing and Optimization Group (PCOG),
University of Luxembourg (UL), Luxembourg

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

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

Jan. 23th, 2018, Zagreb, Croatia

About me

https://varrette.gforge.uni.lu

- 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
  - ↪ $\simeq$ 206.772 **T**Flops (2017), 7952.4 **T**B
  - ↪ 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!

- 3rd 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`

Agenda: Jan. 23th, 2018

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

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

Time	Session
09:00 - 09:30	Discover the Hands-on tool: Vagrant
09:30 - 10:00	HPC and Big Data (BD): Architectures and Trends
10:00 - 10:30	Interlude: Software Management in HPC systems
10:30 - 11:00	[Big] Data Management in HPC Environment: Overview and Challenges
11:00 - 11:15	**Coffee Break**
11:15 - 12:30	Big Data Analytics with Hadoop & Spark
12:30 - 13:00	Deep Learning Analytics with Tensorflow
13:00	**Lunch**

Summary

Summary

Summary

Tutorial Pre-Requisites / Setup

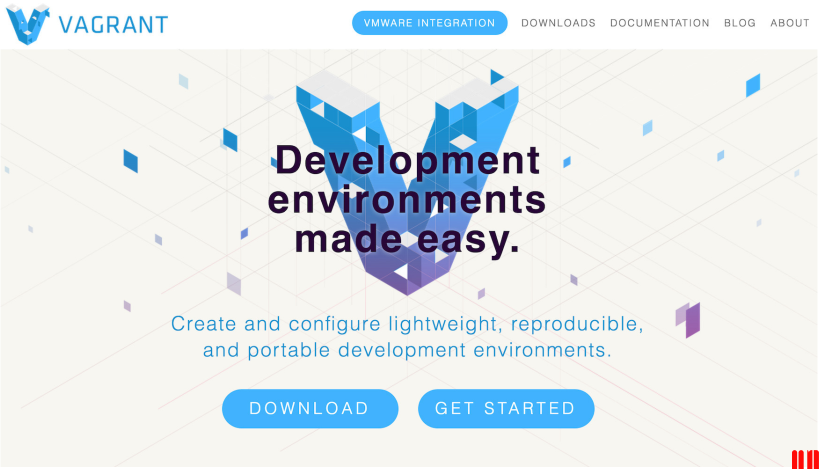
http://nesusws-tutorials-BD-DL.rtfd.io/

- Follow instructions on Getting Started / Pre-requisites
  - ↪ create (if needed) accounts: Github, Vagrant Cloud, Docker Hub
  - ↪ install **mandatory software**, *i.e.* (apart from Git):

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

Discover the Hands-on Tool: Vagrant

http://vagrantup.com/



What is Vagrant ?

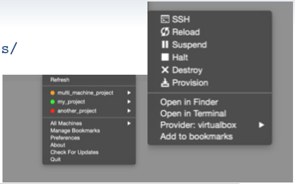
Create and configure **lightweight**, **reproducible**, and **portable** development environments

- **Command line** tool `vagrant [...]`
- 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 https://puppet.com/
  - ↪ Salt... https://saltstack.com/

Cross-platform: runs on Linux, Windows, MacOS

Installation Notes

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/setup/preliminaries/



- **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**:

```
$> 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/
  - ↪ You can have multiple (named) box within the **same** Vagrantfile
    - ✓ See `ULHPC/puppet-sysadmins/Vagrantfile`
    - ✓ See `Falkor/tutorials-BD-ML/Vagrantfile`

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

Box name	Description
ubuntu/trusty64	Ubuntu Server 14.04 LTS
debian/contrib-jessie64	Vanilla Debian 8 Jessie
centos/7	CentOS Linux 7 x86_64

Pulling and Running a Vagrant Box

```
$> 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`

Pulling and Running a Vagrant Box

```
$> 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)
```

Pulling and Running a Vagrant Box

```
$> 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)
```

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

Stopping Vagrant Box

```
$> 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

Hands-on 0: Vagrant

- This tutorial heavily relies on Vagrant
  ↪ you will need to familiarize with the tool if not yet done

Your Turn!

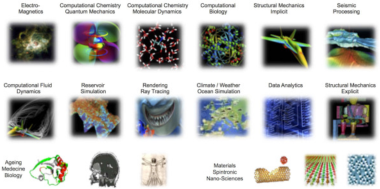
Hands-on 0

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/vagrant/

- **Clone** the tutorial repository — Step 1
- **Basic Usage of Vagrant** — Step 2

Summary

Why HPC and BD ?



HPC: **H**igh **P**erformance **C**omputing

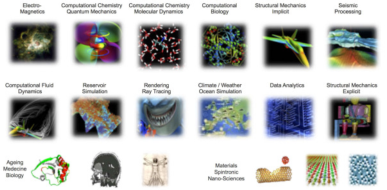
BD: **B**ig **D**ata

Andy Grant, Head of Big Data and HPC, Atos UK&I

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.

Why HPC and BD ?



HPC: **H**igh **P**erformance **C**omputing
BD: **B**ig **D**ata

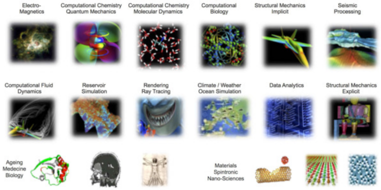
- Essential tools for **Science, Society and Industry**
  - ↪ All scientific disciplines are becoming computational today
    - ✓ requires very high computing power, handles **huge** volumes of data
- **Industry, SMEs** increasingly relying on HPC
  - ↪ to invent innovative solutions
  - ↪ . . . while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

Andy Grant, Head of Big Data and HPC, Atos UK&I

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.

Why HPC and BD ?



HPC: **H**igh **P**erformance **C**omputing
BD: **B**ig **D**ata

- 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

Andy Grant, Head of Big Data and HPC, Atos UK&I

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.

New Trends in HPC

- **Continued scaling** of scientific, industrial & financial applications
  - ↪ . . . well beyond Exascale
- New trends changing the landscape for HPC
  - ↪ Emergence of **Big Data analytics**
  - ↪ Emergence of (**Hyperscale**) **Cloud Computing**
  - ↪ **Data intensive Internet of Things (IoT)** applications
  - ↪ **Deep learning & cognitive computing** paradigms

[Source : EuroLab-4-HPC]


This study was carried out for RIKEN by

IDC Analyze the Future

Special Study

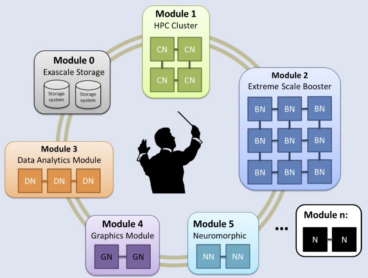
Analysis of the Characteristics and Development Trends of the Next-Generation of Supercomputers in Foreign Countries

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

[Source : IDC RIKEN report, 2016]

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]

Prerequisites: Metrics

- **HPC**: **H**igh **P**erformance **C**omputing — **BD**: **B**ig **D**ata

Main HPC/BD Performance Metrics

- **Computing Capacity**: often measured in **flops** (or **flop/s**)
  - $\hookrightarrow$ **Fl**oating **p**oint **o**perations per **s**econds (often in DP)
  - $\hookrightarrow$ **GFlops** $= 10^9$ **TFlops** $= 10^{12}$ **PFlops** $= 10^{15}$ **EFlops** $= 10^{18}$

Prerequisites: Metrics

- **HPC**: **H**igh **P**erformance **C**omputing **BD**: **B**ig **D**ata

Main HPC/BD Performance Metrics

- **Computing Capacity**: often measured in **flops** (or **flop/s**)
  - ↪ **F**loating **p**oint **o**perations per **s**econds (often in DP)
  - ↪ **GFlops** $= 10^9$ **TFlops** $= 10^{12}$ **PFlops** $= 10^{15}$ **EFlops** $= 10^{18}$
- **Storage Capacity**: measured in multiples of **bytes** = 8 **bits**
  - ↪ **GB** $= 10^9$ bytes **TB** $= 10^{12}$ **PB** $= 10^{15}$ **EB** $= 10^{18}$
  - ↪ **GiB** $= 1024^3$ bytes **TiB** $= 1024^4$ **PiB** $= 1024^5$ **EiB** $= 1024^6$
- Transfer rate on a medium measured in **Mb/s** or **MB/s**
- Other metrics: Sequential vs Random **R/W speed**, **IOPS** ...

Summary

HPC Components: [GP]CPU

CPU

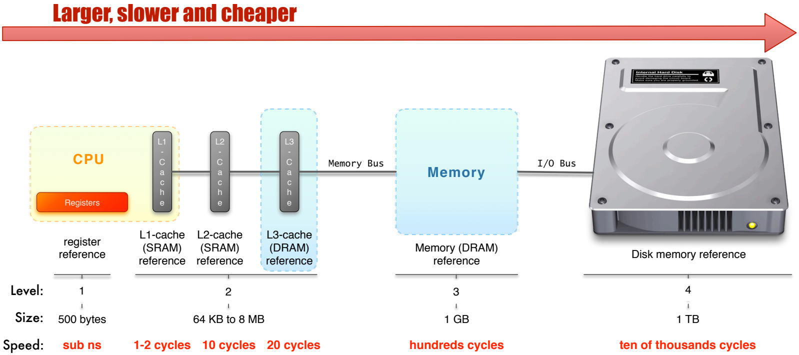
- Always multi-core
- Ex: Intel Core i7-7700K (Jan 2017) $R_{peak} \simeq 268.8$ GFlops (DP)
  - ↪ 4 cores @ 4.2GHz (14nm, 91W, 1.75 billion transistors)
  - ↪ + integrated graphics (24 EUs) $R_{peak} \simeq +441.6$ GFlops

GPU / GPGPU

- Always multi-core, optimized for vector processing
- Ex: Nvidia Tesla V100 (Jun 2017) $R_{peak} \simeq 7$ TFlops (DP)
  - ↪ 5120 cores @ 1.3GHz (12nm, 250W, 21 billion transistors)
  - ↪ focus on Deep Learning workloads $R_{peak} \simeq 112$ TFLOPS (HP)

$\simeq$ **100 Gflops** for 130$ (CPU), 214$? (GPU)

HPC Components: Local Memory

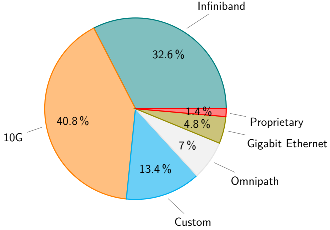


- SSD (SATA3) R/W: 550 MB/s; 100000 IOPS **450 €/TB**
- HDD (SATA3 @ 7,2 krpm) R/W: 227 MB/s; 85 IOPS **54 €/TB**

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

Technology	Effective Bandwidth		Latency
Gigabit Ethernet	1 Gb/s	125 MB/s	40μs to 300μs
10 Gigabit Ethernet	10 Gb/s	1.25 GB/s	4μs to 5μs
Infiniband QDR	40 Gb/s	5 GB/s	1.29μs to 2.6μs
Infiniband EDR	100 Gb/s	12.5 GB/s	0.61μs to 1.3μs
100 Gigabit Ethernet	100 Gb/s	1.25 GB/s	30μs
Intel Omnipath	100 Gb/s	12.5 GB/s	0.9μs

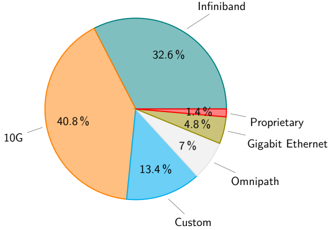


[Source : www.top500.org, Nov. 2017]

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

Technology	Effective Bandwidth		Latency
Gigabit Ethernet	1 Gb/s	125 MB/s	40μs to 300μs
10 Gigabit Ethernet	10 Gb/s	1.25 GB/s	4μs to 5μs
Infiniband QDR	40 Gb/s	5 GB/s	1.29μs to 2.6μs
Infiniband EDR	100 Gb/s	12.5 GB/s	0.61μs to 1.3μs
100 Gigabit Ethernet	100 Gb/s	1.25 GB/s	30μs
Intel Omnipath	100 Gb/s	12.5 GB/s	0.9μs



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

Network Topologies

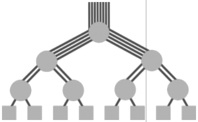
- **Direct** vs. **Indirect** interconnect
  - ↪ *direct*: each network node attaches to at least one compute node
  - ↪ *indirect*: compute nodes attached at the edge of the network only
    - ✓ many routers only connect to other routers.

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

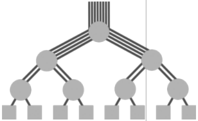
- **CLOS Network / Fat-Trees** [Indirect]
  - ↪ can be fully non-blocking (1:1) or blocking (x:1)
  - ↪ typically enables **best performance**
    - ✓ Non blocking bandwidth, lowest network latency



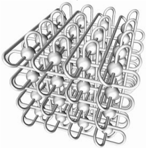
Network Topologies

- **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

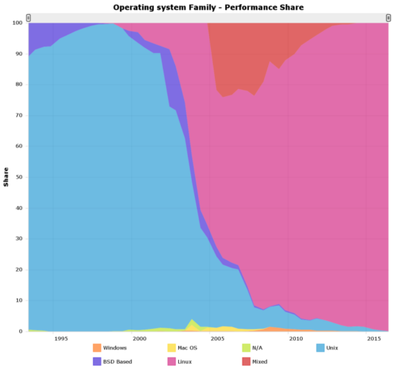


- **CLOS Network / Fat-Trees** [Indirect]
  - ↪ can be fully non-blocking (1:1) or blocking (x:1)
  - ↪ typically enables **best performance**
    - ✓ Non blocking bandwidth, lowest network latency



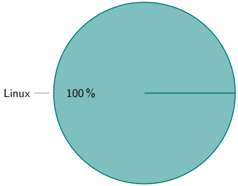
- **Mesh or 3D-torus** [Direct]
  - ↪ Blocking network, cost-effective for systems at scale
  - ↪ Great performance solutions for applications with locality
  - ↪ Simple expansion for future growth

HPC Components: Operating System



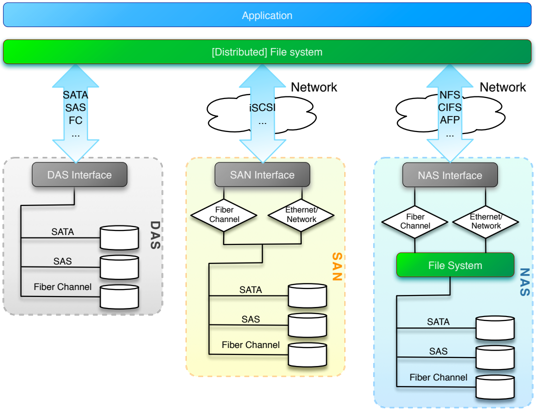
- Exclusively Linux-based (**really** 100%)
- Reasons:
  - ↪ stability
  - ↪ prone to devels

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



[Big]Data Management

Storage architectural classes & I/O layers



[Big]Data Management: Disk Encl.

- $\simeq$ **120 K€** - enclosure - 48-60 disks (4U)
  - $\hookrightarrow$ incl. redundant (**i.e.** 2) RAID controllers (master/slave)

[Big]Data Management: File Systems

File System (FS)

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

[Big]Data Management: File Systems

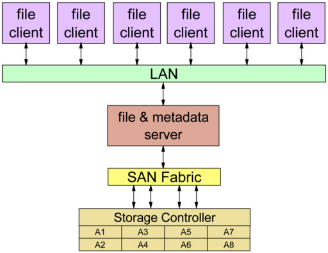
File System (FS)

- 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

[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)

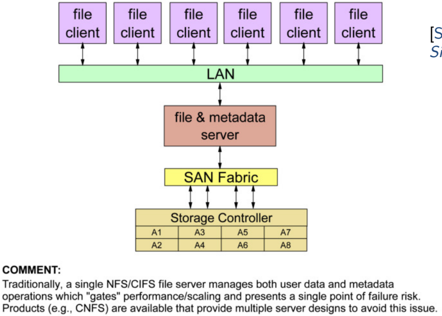


[Source : LISA'09] Ray Paden: *How to Build a Petabyte Sized Storage System*

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)

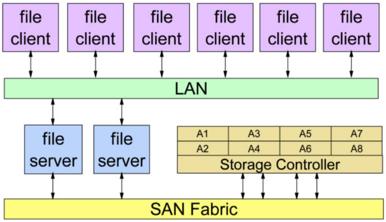


[Source : LISA'09] Ray Paden: *How to Build a Petabyte Sized Storage System*

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

[Big]Data Management: File Systems

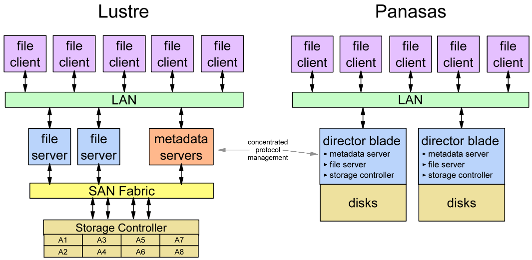
- **Basic Clustered FS** GPFS
  - ↪ File access is parallel
  - ↪ File System overhead operations is distributed and done in parallel
    - ✓ **no** metadata servers
  - ↪ File clients access file data through file servers via the LAN



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 betweem all nodes and all disks via the file servers.

[Big]Data Management: File Systems

- **Multi-Component Clustered FS** Lustre, Panasas
  - ↪ 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



[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)

[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

[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

Name	Type	Read* [GB/s]	Write* [GB/s]
`ext4`	Disk FS	0.426	0.212
`nfs`	Networked FS	0.381	0.090
`gpfs` (iris)	Parallel/Distributed FS	**10.14**	**8,41**
`gpfs` (gaia)	Parallel/Distributed FS	7.74	6.524
`lustre`	Parallel/Distributed FS	4.5	2.956

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

HPC Components: Data Center

Definition (Data Center)

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

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: **15 kW**
  - ↪ **Interconnect** racks: **5 kW**
- Various **Cooling** Technology
  - ↪ Airflow
  - ↪ Direct-Liquid Cooling, Immersion...

Power Usage Effectiveness

$$PUE = \frac{\textit{Total facility power}}{\textit{IT equipment power}}$$

Summary

Software/Modules Management

https://hpc.uni.lu/users/software/

- 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}`

```
$> module avail                        # List available modules
```

```
$> module load <category>/<software>[/<version>]
```

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
```

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:

Command	Description
`module avail`	Lists all the modules which are available to be loaded
`module spider <pattern>`	Search for among available modules **(Lmod only)**
`module load <mod1> [mod2...]`	Load a module
`module unload <module>`	Unload a module
`module list`	List loaded modules
`module purge`	Unload all modules (purge)
`module display <module>`	Display what a module does
`module use <path>`	Prepend the directory to the MODULEPATH environment variable
`module unuse <path>`	Remove the directory from the MODULEPATH environment variable

Software/Modules Management

http://hpcugent.github.io/easybuild/

- 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**

```
$> 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
```

Hands-on 1: Modules & Easybuild

Your Turn!

Hands-on 1

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/easybuild/

- **Discover** Environment Modules and Lmod — Part 1
- Installation of EasyBuild — Part 2 (a)
- **Local** vs. **Global** Usage — Part 2 (b)
  - ↪ local installation of zlib
  - ↪ global installation of snappy and protobuf, needed later

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

`http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/hadoop/install/`

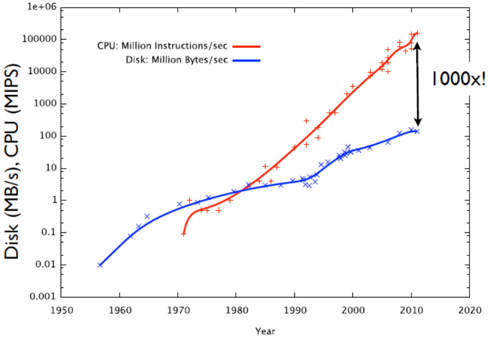
- Pre-requisites — Step 1
  - ↪ Installing **Java 1.7.0** (7u80) and **1.8.0** (8u152) — Step 1.a
  - ↪ Installing **Maven 3.5.2** — Step 1.b
- Installing **Hadoop 2.6.0-cdh5.12.0** — Step 2

Summary

Summary

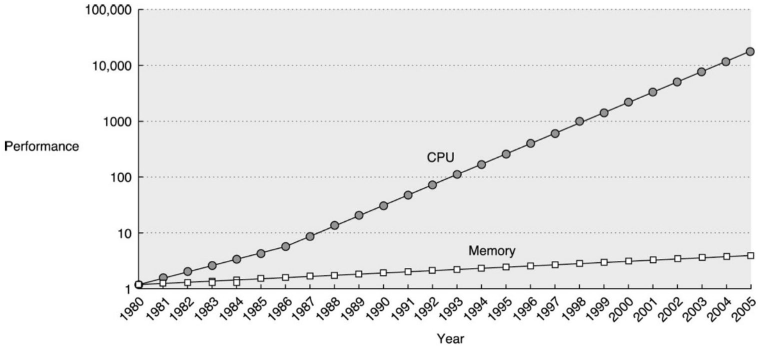
Data Intensive Computing

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



Data Intensive Computing

- 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

http://fasterdata.es.net/

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

Network	Time
10 Mbps	300 hrs (12.5 days)
100 Mbps	30 hrs
1 Gbps	3 hrs
10 Gbps	20 minutes

- **(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

http://fasterdata.es.net/

Data set size				
10PB	166.67 TB/sec	33.33 TB/sec	8.33 TB/sec	2.78 TB/sec
1PB	16.67 TB/sec	3.33 TB/sec	833.33 GB/sec	277.78 GB/sec
100TB	1.67 TB/sec	333.33 GB/sec	83.33 GB/sec	27.78 GB/sec
10TB	166.67 GB/sec	33.33 GB/sec	8.33 GB/sec	2.78 GB/sec
1TB	16.67 GB/sec	3.33 GB/sec	833.33 MB/sec	277.78 MB/sec
100GB	1.67 GB/sec	333.33 MB/sec	83.33 MB/sec	27.78 MB/sec
10GB	166.67 MB/sec	33.33 MB/sec	8.33 MB/sec	2.78 MB/sec
1GB	16.67 MB/sec	3.33 MB/sec	0.83 MB/sec	0.28 MB/sec
100MB	1.67 MB/sec	0.33 MB/sec	0.08 MB/sec	0.03 MB/sec
	1 Minute	5 Minutes	20 Minutes	1 Hour
	Time to transfer			

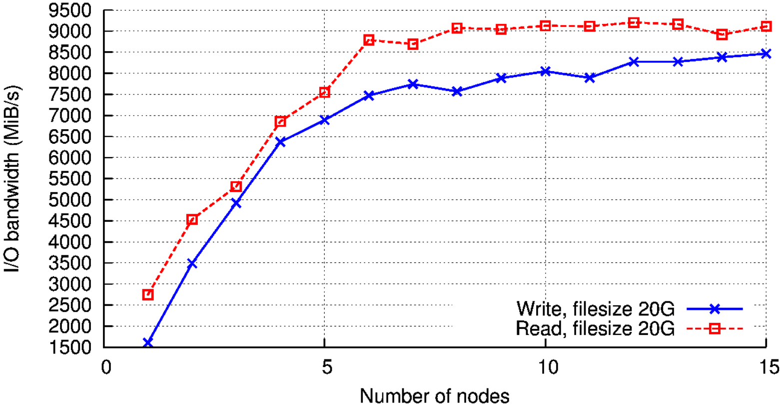
Speed Expectation on Data Transfer

http://fasterdata.es.net/

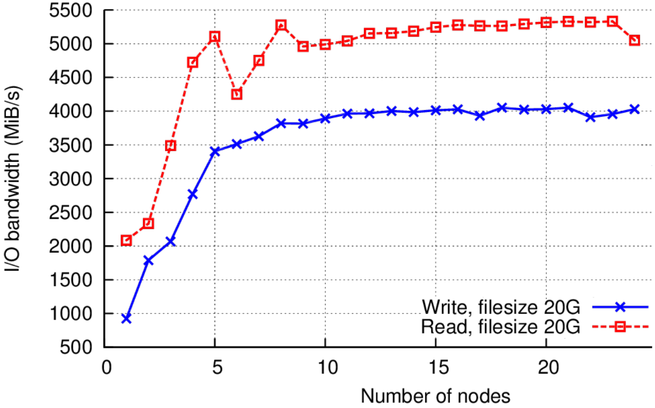
Data set size	8 Hours	24 Hours	7 Days	30 Days
1XB	34.72 TB/sec	11.57 TB/sec	1.65 TB/sec	385.80 GB/sec
100PB	3.47 TB/sec	1.16 TB/sec	165.34 GB/sec	38.58 GB/sec
10PB	347.22 GB/sec	115.74 GB/sec	16.53 GB/sec	3.86 GB/sec
1PB	34.72 GB/sec	11.57 GB/sec	1.65 GB/sec	385.80 MB/sec
100TB	3.47 GB/sec	1.16 GB/sec	165.34 MB/sec	38.58 MB/sec
10TB	347.22 MB/sec	115.74 MB/sec	16.53 MB/sec	3.86 MB/sec
1TB	34.72 MB/sec	11.57 MB/sec	1.65 MB/sec	0.39 MB/sec
100GB	3.47 MB/sec	1.16 MB/sec	0.17 MB/sec	0.04 MB/sec
10GB	0.35 MB/sec	0.12 MB/sec	0.02 MB/sec	0.00 MB/sec

Time to transfer

Storage Performances: GPFS

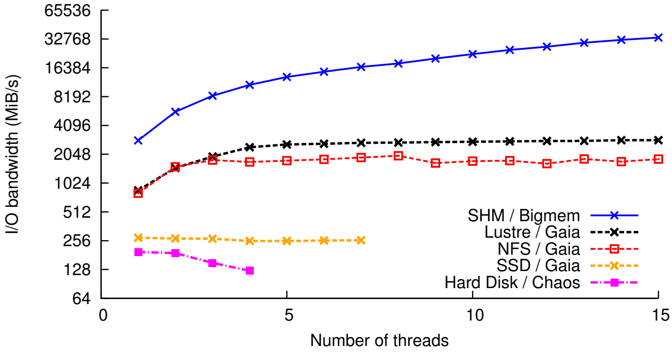


Storage Performances: Lustre



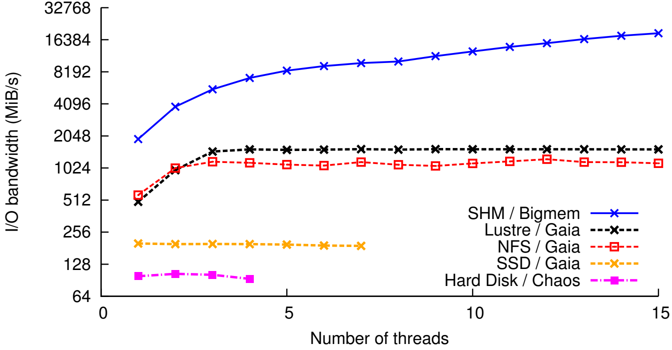
Storage Performances

- Based on IOR or IOZone, reference I/O benchmarks **Read**
  ↪ *tests performed in 2013*



Storage Performances

- Based on IOR or IOZone, reference I/O benchmarks **Write**
  ↪ *tests performed in 2013*



Understanding Your Storage Options

Where can I store and manipulate my data?

- **Shared** storage
  - ↪ `NFS` - **not scalable** $\sim\simeq$ 1.5 GB/s (R) — $\mathcal{O}$(100 TB)
  - ↪ `GPFS` - **scalable** $\sim\sim\simeq$ 10 GB/s (R) — $\mathcal{O}$(1 PB)
  - ↪ `Lustre` - **scalable** $\sim\sim\simeq$ 5 GB/s (R) — $\mathcal{O}$(0.5 PB)
- **Local** storage
  - ↪ local file system (`/tmp`) — $\mathcal{O}$(200 GB)
    - ✓ over HDD $\simeq$ 100 MB/s, over SDD $\simeq$ 400 MB/s
  - ↪ RAM (`/dev/shm`) $\simeq$ 30 GB/s (R) — $\mathcal{O}$(20 GB)
- **Distributed** storage
  - ↪ HDFS, Ceph, GlusterFS - **scalable** $\sim\sim\simeq$ 1 GB/s

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

Summary

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] `wget` or (better) `curl`
  - ↪ can also serve to send HTTP POST requests
  - ↪ support HTTP cookies (useful for JDK download)

Data Transfer in Practice

```
$> scp [-P <port>] <src> <user>@<host>:<path>
```

```
$> rsync -avzu [-e 'ssh -p <port>'] <src> <user>@<host>:<path>
```

- [Secure] Transfer **from/to** two remote machines over SSH
  ↪ `scp` or (better) `rsync` (transfer **only** 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

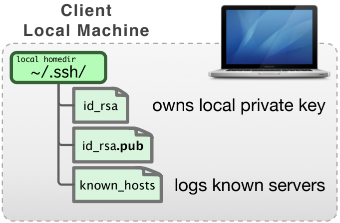
SSH: Secure Shell

- 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.

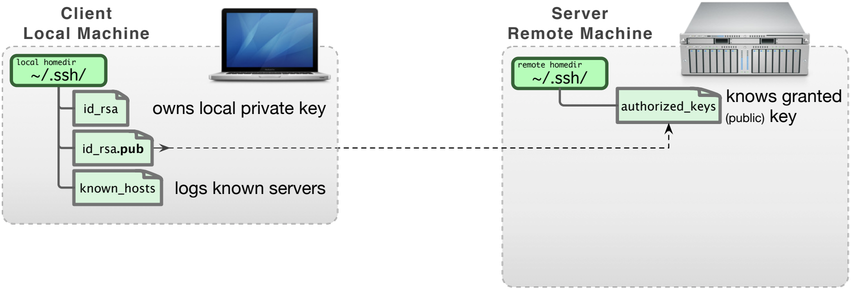
SSH: Secure Shell

- 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**

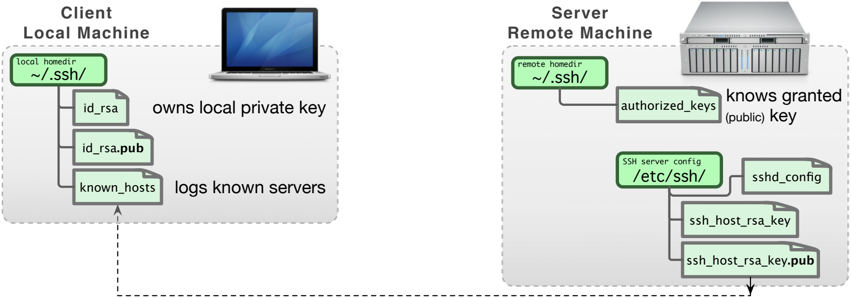
SSH: Public Key Authentication



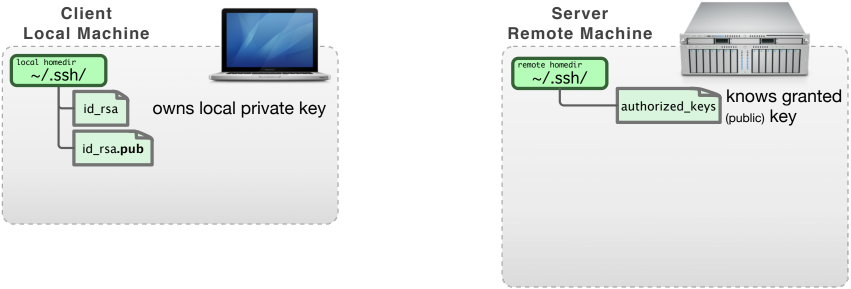
SSH: Public Key Authentication



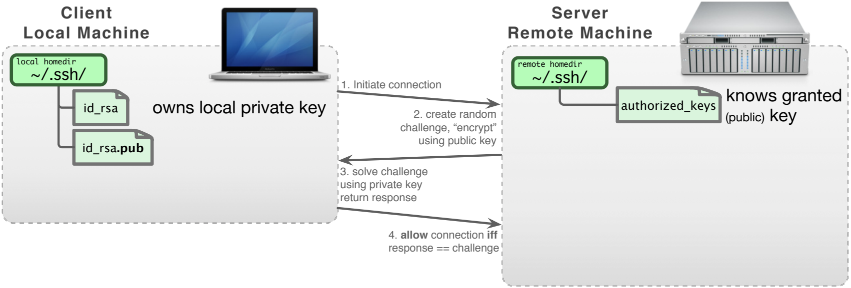
SSH: Public Key Authentication



SSH: Public Key Authentication



SSH: Public Key Authentication



- **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 transfert it between the 2 Vagrant boxes ;)
```

Hands-on 3

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/data-transfer/

- **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

Sharing Code and Data

- 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

Sharing Code and Data

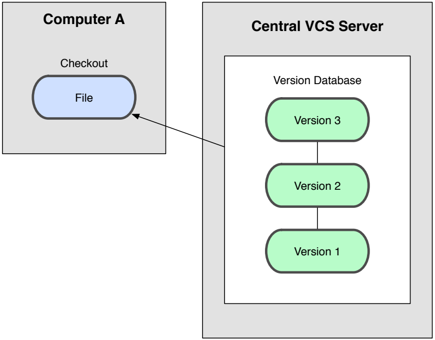
- Before doing **Big** Data, manage and version correctly **normal** data

What kinds of systems are available?

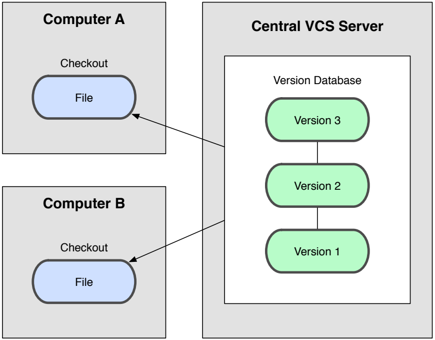
- *Good*: NAS, Cloud Dropbox, Google Drive, Figshare...
- **Better - Version Control systems** (VCS)
  ↪ SVN, Git and Mercurial
- **Best - Version Control Systems** on the **Public/Private Cloud**
  ↪ GitHub, Bitbucket, Gitlab

- **Which one?**
  ↪ Depends on the level of privacy you expect
  ✓ ... but you probably already know these tools ☺
  ↪ **Few handle GB files...**

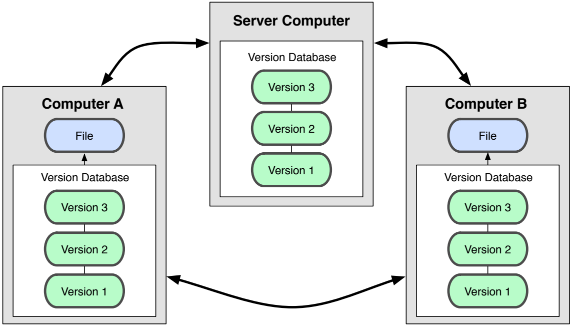
Centralized VCS - CVS, SVN



Centralized VCS - CVS, SVN

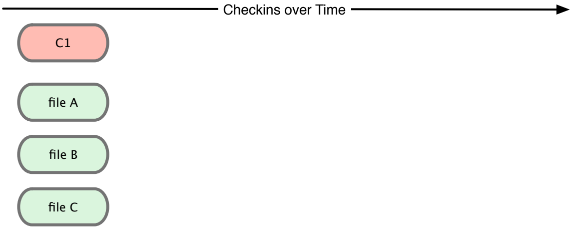


Distributed VCS - Git

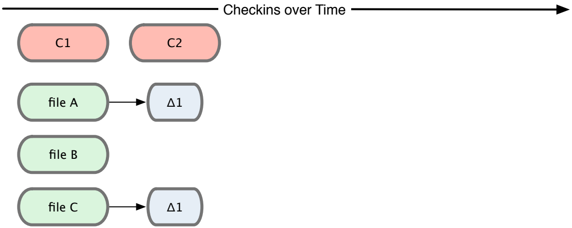


Everybody has the full history of commits

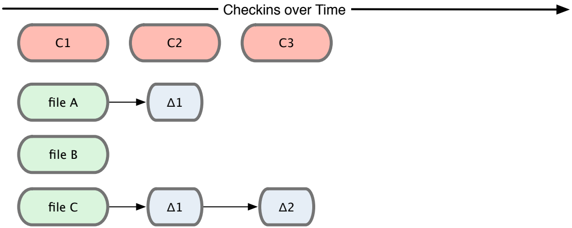
Tracking changes (most VCS)



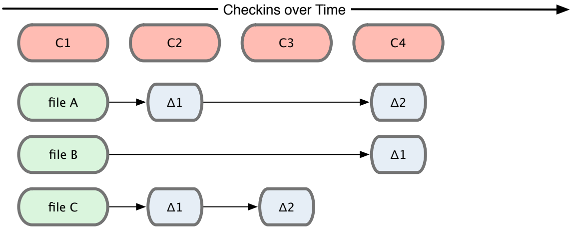
Tracking changes (most VCS)



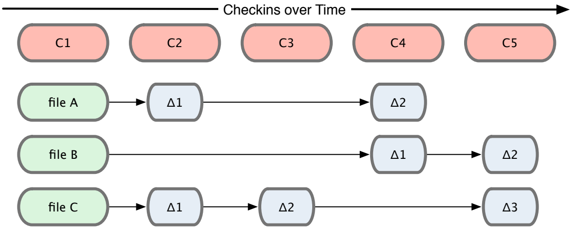
Tracking changes (most VCS)



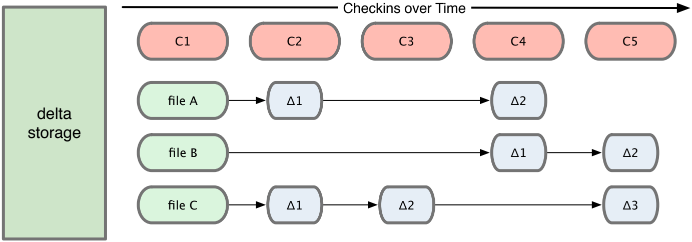
Tracking changes (most VCS)



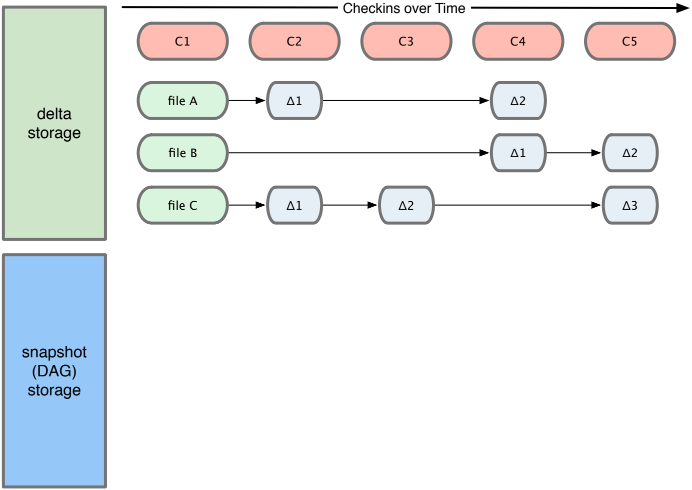
Tracking changes (most VCS)



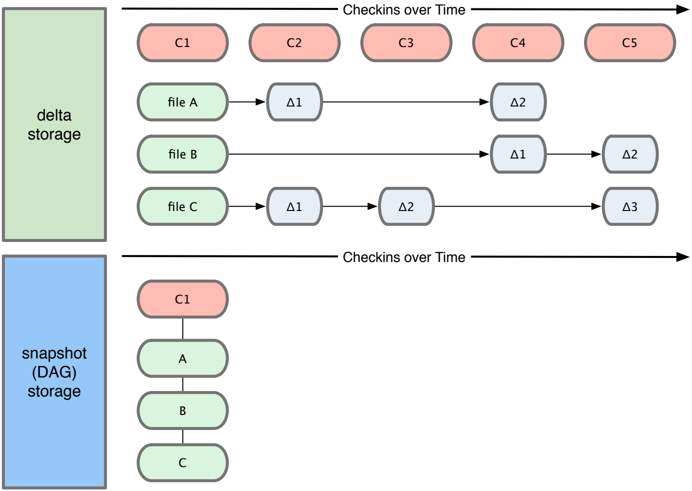
Tracking changes (most VCS)



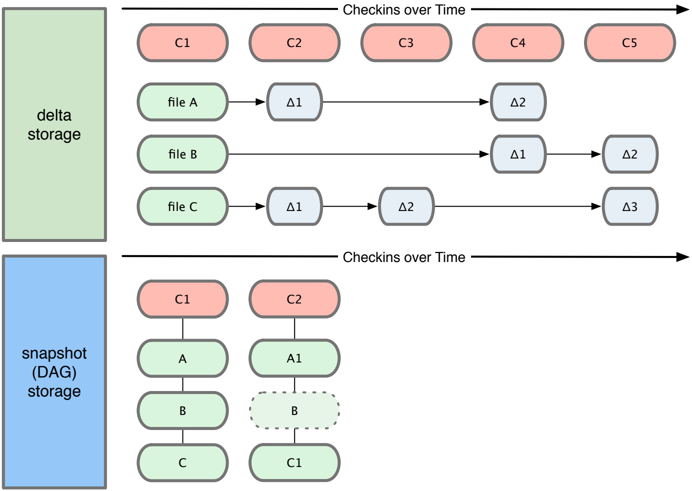
Tracking changes (Git)



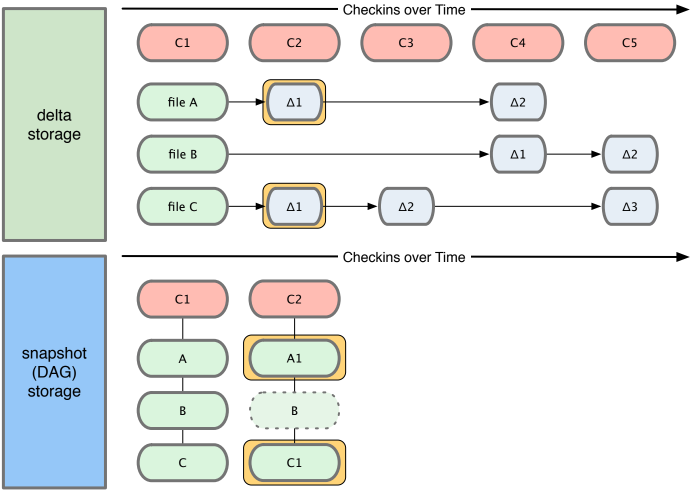
Tracking changes (Git)



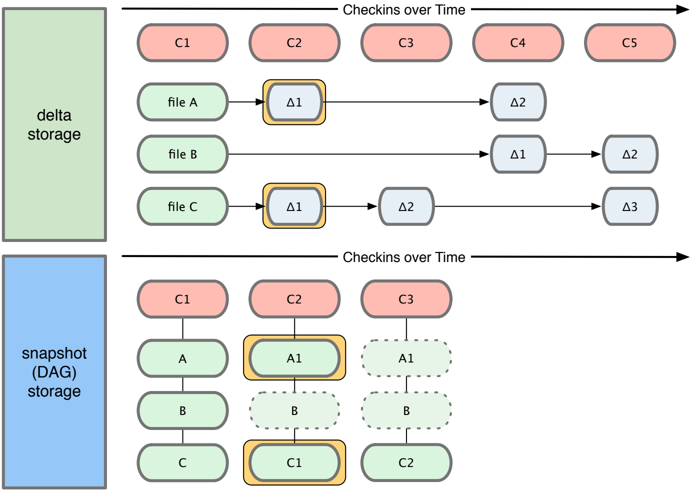
Tracking changes (Git)



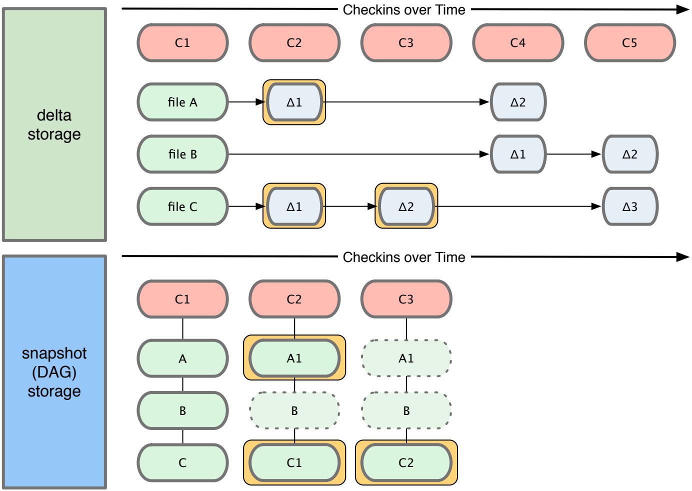
Tracking changes (Git)



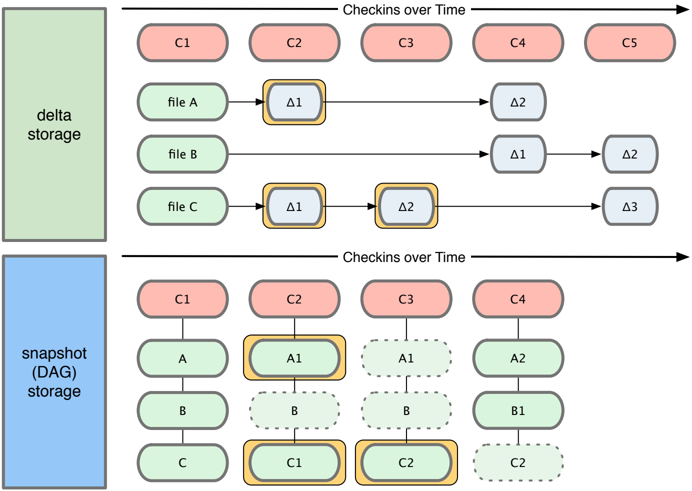
Tracking changes (Git)



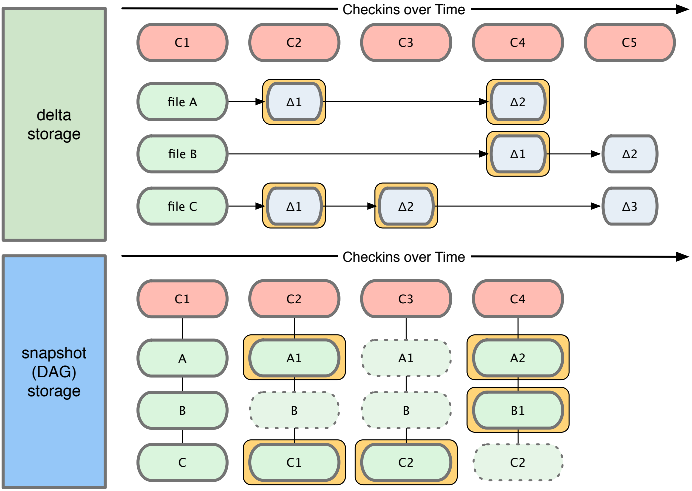
Tracking changes (Git)



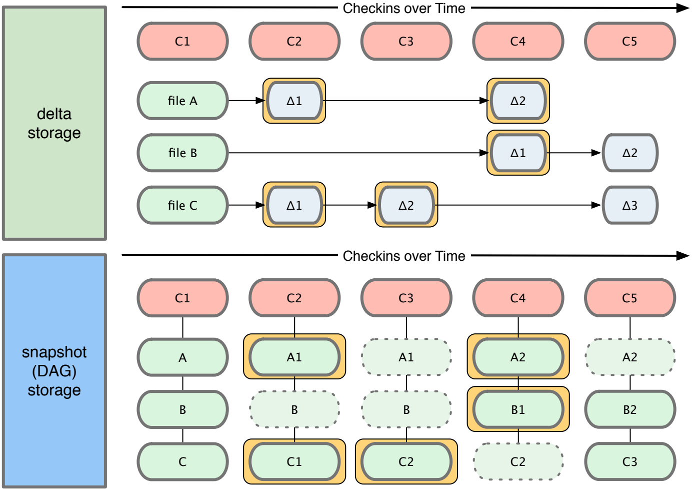
Tracking changes (Git)



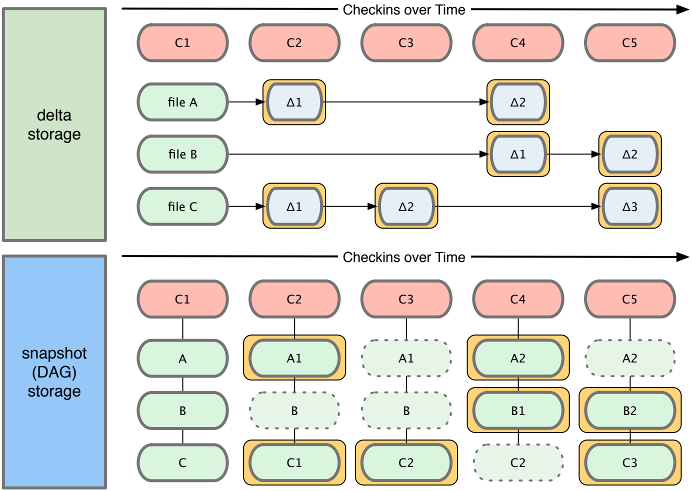
Tracking changes (Git)



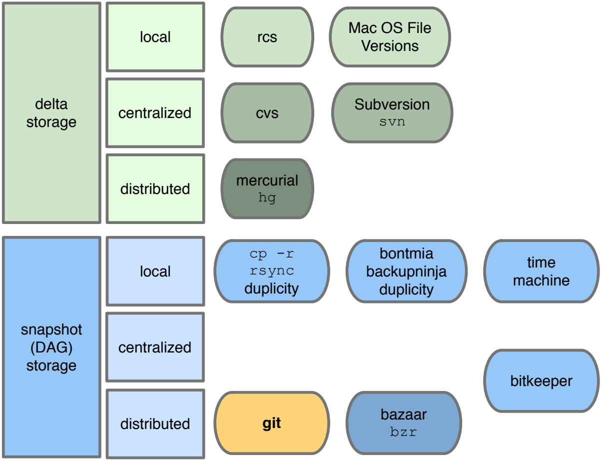
Tracking changes (Git)



Tracking changes (Git)



VCS Taxonomy



Git at the heart of BD

http://git-scm.org

Git on the Cloud: Github `github.com`

(Reference) web-based Git repository hosting service

Set up Git

Create Repository

Fork repository

Work together

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**
  $\hookrightarrow$ 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

```
$> 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	`git pull`
• **Edit** files	`vim` / `emacs` / `subl` ...
• **Stage** the changes	`git add`
• **Review** your changes	`git status`
• **Commit** the changes	`git commit`

Git in practice

Basic Workflow

- **Pull** latest changes — `git pull`
- **Edit** files — `vim` / `emacs` / `subl` ...
- **Stage** the changes — `git add`
- **Review** your changes — `git status`
- **Commit** the changes — `git commit`

For cheaters: A Basicerer Workflow

- **Pull** latest changes — `git pull`
- **Edit** files — `vim` / `emacs` / `subl` ...
- Stage & commit all the changes — `git commit -a`

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: IT/Dev[op]s Army Knives Tools for the Researcher
    - ✓ tutorial: Reproducible Research at the Cloud Era

Summary

Summary

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

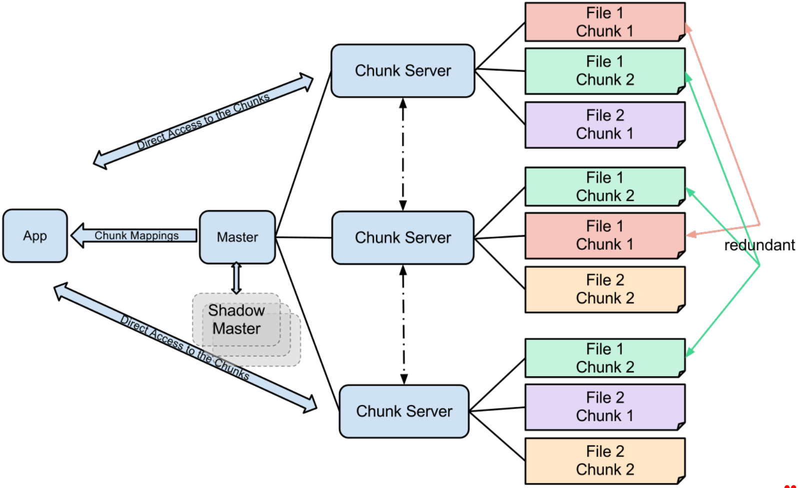
Peer-to-Peer Pattern

- No global state information.
- Each node may both serve and process data.

Google File System (GFS) (2003)

- 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

Google File System (GFS) (2003)



GFS Consistency Model

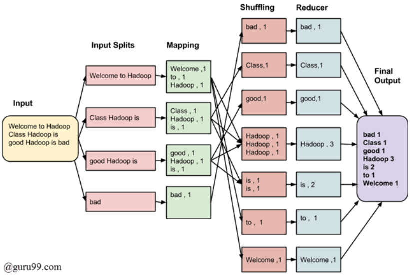
- **Atomic File Namespace Mutations**
  - ↪ File creations/deletions centrally controlled by the master node.
  - ↪ Clients typically create and write entire file,
    - ✓ then add the file name to the file namespace stored at the master.
- **Atomic Data Mutations**
  - ↪ only 1 atomic modification of 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



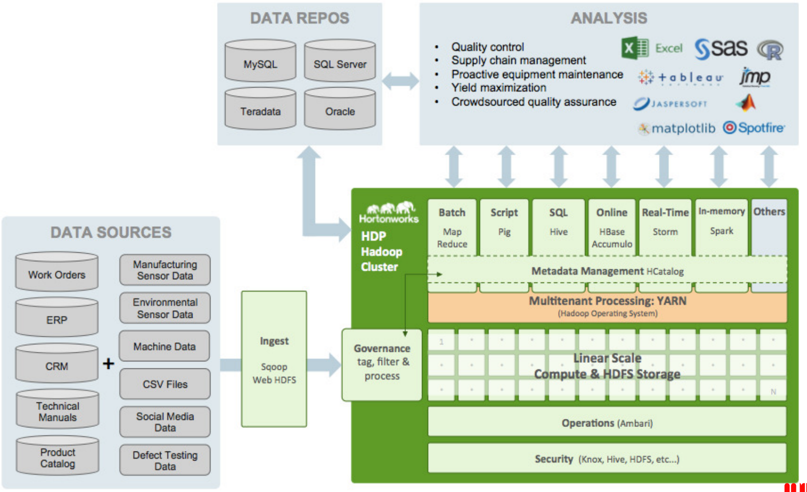
Hadoop

- 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

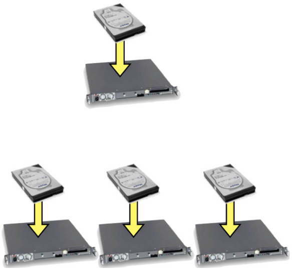
Hadoop

- 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.

Hadoop Ecosystem

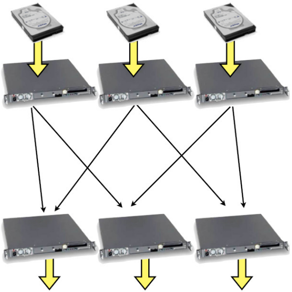


Scale-Out Design



- HDD streaming speed $\sim$ 50MB/s
  - ↪ 3TB =17.5 hrs
  - ↪ 1PB = 8 months
- Scale-out (weak scaling)
  - ↪ **FS distributes data on ingest**
- Seeking too slow
  - ↪ ~10ms for a seek
  - ↪ Enough time to read half a megabyte
- **Batch processing**
- Go through entire data set in one (or small number) of passes

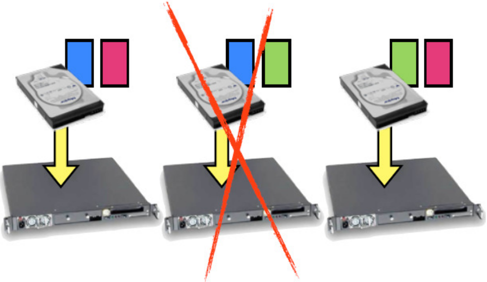
Combining Results



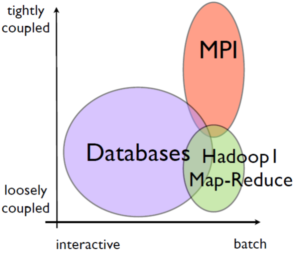
- Each node preprocesses its local data
  - ↪ Shuffles its data to a small number of other nodes
- Final processing, output is done there

Fault Tolerance

- Data also replicated upon ingest
- Runtime watches for dead tasks, restarts them on live nodes
- Re-replicates

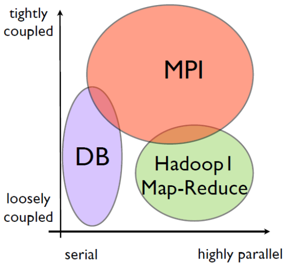


Hadoop: What is it Good At?



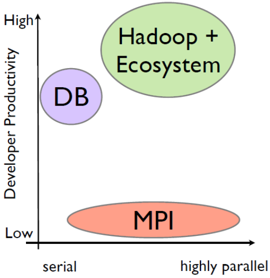
- **Classic Hadoop 1.x is all about batch processing** of massive amounts of data
  - ↪ Not much point below ~1TB
- Map-Reduce is relatively loosely coupled;
  - ↪ one **shuffle** phase.
- Very strong weak scaling in this model
  - ↪ more data, more nodes.
- **Batch**:
  - ↪ process all data in one go
    - ✓ 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.

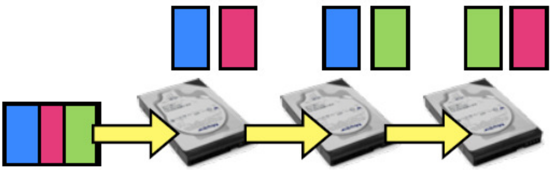
Hadoop vs HPC



- 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? Oooooooh*
- **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...

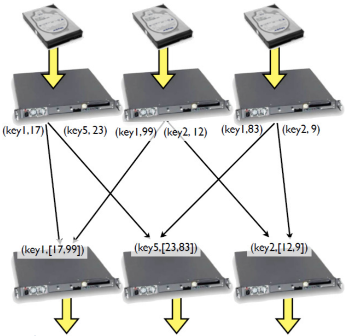
Data Distribution: Disk

- 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



Data Distribution: Network

- **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



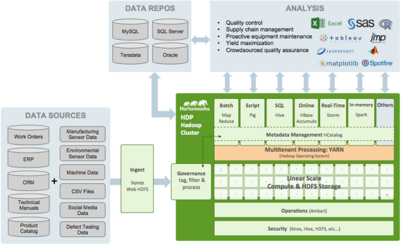
Makes the problem easier

- **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.

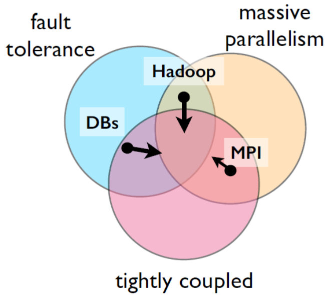
Built a reusable substrate

- 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



The Hadoop Filesystem

- **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]
```

cat	chgrp
chmod	chown
copyFromLocal	copyToLocal
cp	du
dus	expunge
get	getmerge
ls	lsr
mkdir	movefromLocal
mv	put
rm	rmr
setrep	stat
tail	test
text	touchz

The Hadoop Filesystem

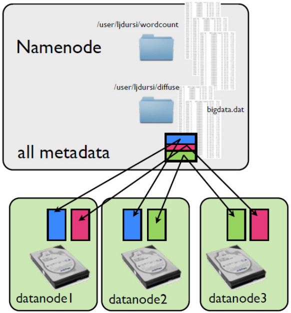
- **Required** to be:
  - ↪ able to deal with large files, large amounts of data
  - ↪ scalable & reliable in the presence of failures
  - ↪ fast at reading contiguous streams of data
  - ↪ only need to write to new files or append to files
  - ↪ require only commodity hardware
- **As a result**:
  -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

Datanodes and Namenode

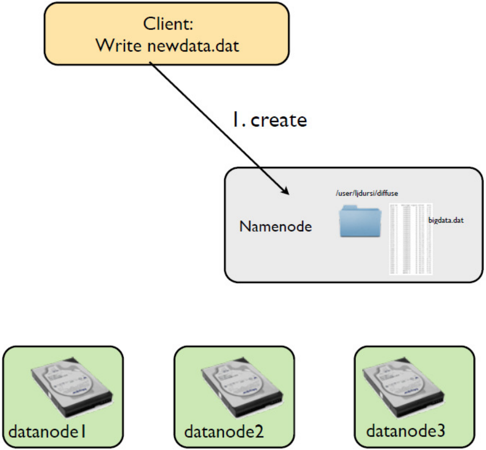


Two types of nodes in the filesystem:

1. **Namenode**
   - ↪ stores all metadata / block locations in memory
   - ↪ Metadata updates stored to persistent journal
2. **Datanodes**
   - ↪ store/retrieve blocks for client/namenode

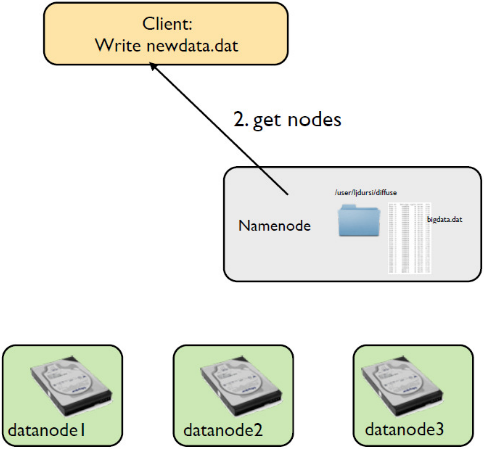
- Newer versions of Hadoop: federation
  - ↪ $\neq$ namenodes for `/user`, `/data`...
  - ↪ High Availability namenode pairs

Writing a file



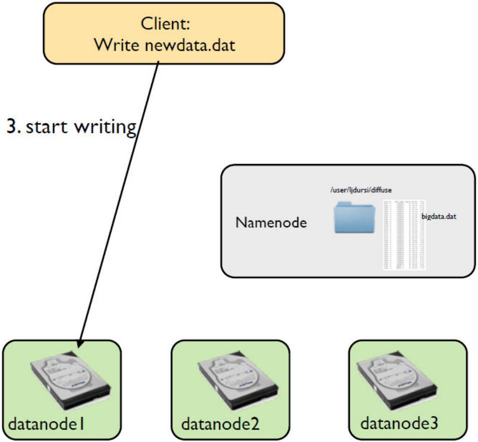
- **Writing a file** multiple stage process:
  - ↪ Create file
  - ↪ Get nodes for blocks
  - ↪ Start writing
  - ↪ Data nodes coordinate replication
  - ↪ Get ack back
  - ↪ Complete

Writing a file



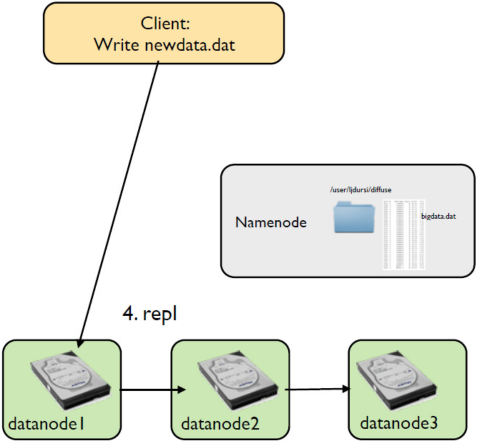
- **Writing a file** multiple stage process:
  - ↪ Create file
  - ↪ Get nodes for blocks
  - ↪ Start writing
  - ↪ Data nodes coordinate replication
  - ↪ Get ack back
  - ↪ Complete

Writing a file



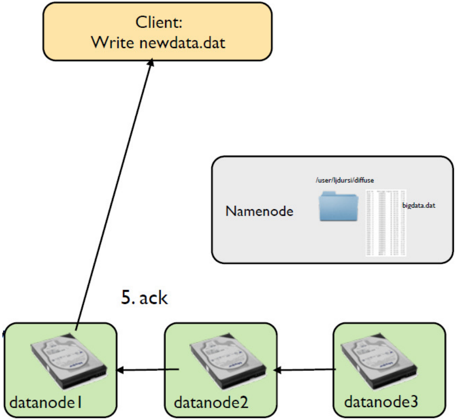
- **Writing a file** multiple stage process:
  - ↪ Create file
  - ↪ Get nodes for blocks
  - ↪ Start writing
  - ↪ Data nodes coordinate replication
  - ↪ Get ack back
  - ↪ Complete

Writing a file



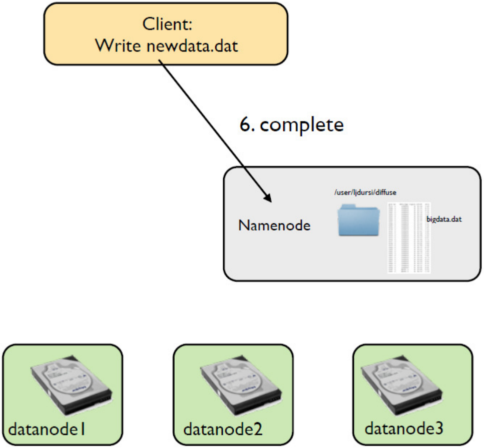
- **Writing a file** multiple stage process:
  - ↪ Create file
  - ↪ Get nodes for blocks
  - ↪ Start writing
  - ↪ Data nodes coordinate replication
  - ↪ Get ack back
  - ↪ Complete

Writing a file



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

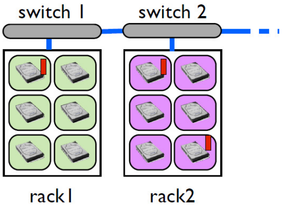
Writing a file



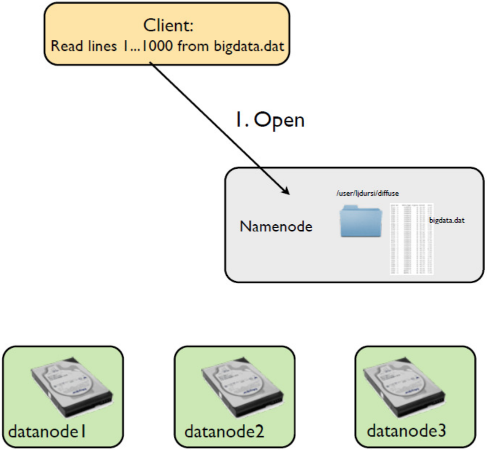
- **Writing a file** multiple stage process:
  - ↪ Create file
  - ↪ Get nodes for blocks
  - ↪ Start writing
  - ↪ Data nodes coordinate replication
  - ↪ Get ack back **(while writing)**
  - ↪ Complete

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

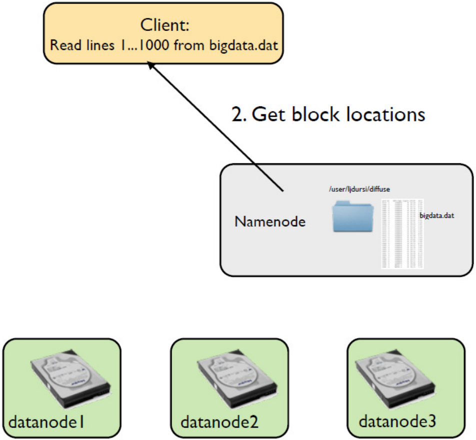


Reading a file



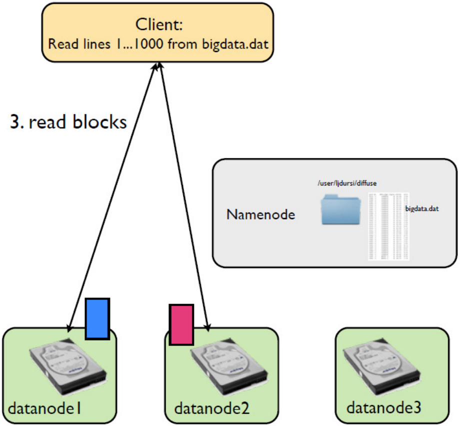
- **Reading** a file
  - ↪ Open call
  - ↪ Get block locations
  - ↪ Read from a replica

Reading a file



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

Reading a file



- **Reading** a file
  - ↪ Open call
  - ↪ Get block locations
  - ↪ 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

Configuring HDFS

```
<!-- $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>
```

Configuring HDFS

- 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

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
[...]
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
```

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 accross 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 accross 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

```
$> hdfs namenode -format
$> start-dfs.sh
```

Using HDFS

- Once the file system is up and running,
  - ↪ . . . you can copy files back and forth

```
$> hadoop fs -{get|put|copyFromLocal|copyToLocal} [...]
```

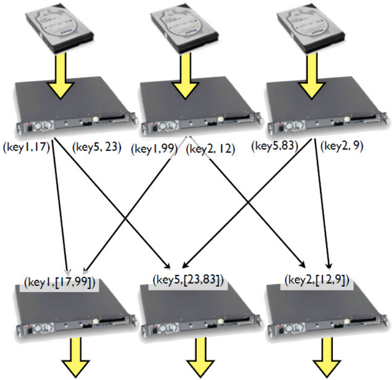
- Default directory is /user/${username}
  - ↪ **Nothing like a cd**

```
$> hdfs fs -mkdir /home/vagrant/hdfs-test
$> hdfs fs -ls    /home/vagrant
$> hdfs fs -ls    /home/vagrant/hdfs-test
$> hdfs fs -put data.dat /home/vagrant/hdfs-test
$> hdfs fs -ls    /home/vagrant/hdfs-test
```

Using HDFS

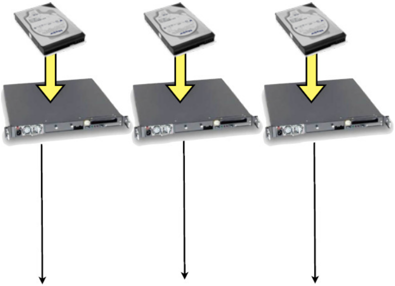
- In general, the data files you send to HDFS will be **large**
  ↪ or else why bother with Hadoop.
- Do not want to be constantly copying back and forth
  ↪ **view, append *in place***
- Several APIs to accessing the HDFS
  ↪ Java, C++, Python
- Here, we use one to get a file status, and read some data from it at some given offset

Back to Map-Reduce



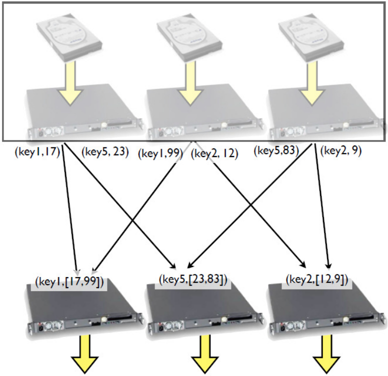
- Map processes **one element at a time**
  - ↪ emits results as (key, value) pairs.
- All results with **same key are gathered to the same reducers**
  - ↪ Reducers process list of values
  - ↪ emit results as (key, value) pairs

Map



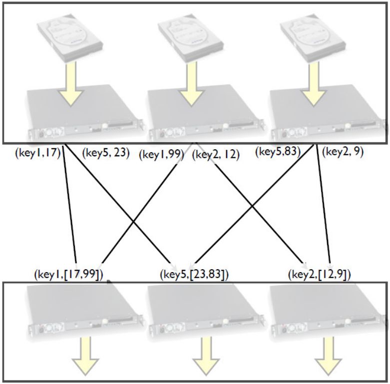
- All coupling done during **shuffle** phase
  - ↪ Embarrassingly parallel task
  - ↪ all map
- Take input, map it to output, done.
- **Famous case**
  - ↪ NYT using Hadoop to convert 11 million image files to PDFs
    - ✓ almost pure serial farm job

Reduce



- 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 occurences of words** within these documents

file01

```
Hello World
Bye World
```

file02

```
Hello Hadoop
Goodbye Hadoop
```

output/part-00000

```
Hello    2
World    2
Bye      1
Hadoop   2
Goodbye  1
```

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

file01

```
Hello World
Bye World
```

file02

```
Hello Hadoop
Goodbye Hadoop
```

⇓

output/part-00000

```
Hello     2
World     2
Bye       1
Hadoop    2
Goodbye   1
```

Example: Wordcount

file01

```
Hello World
Bye World
```

file02

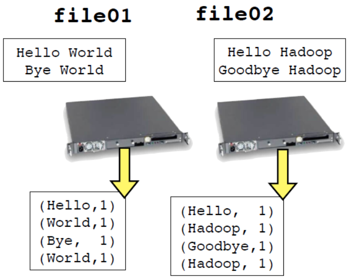
```
Hello Hadoop
Goodbye Hadoop
```

output/part-00000

```
Hello    2
World    2
Bye      1
Hadoop   2
Goodbye  1
```

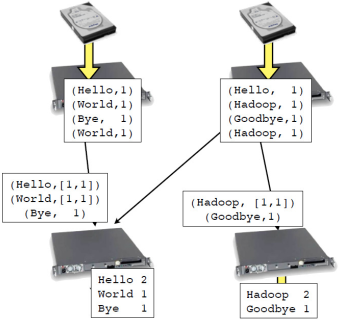
- **MapReduce way**
  - ↪ all hard work done automatically by shuffle
- **Map**:
  - ↪ just emit a 1 for each word you see
- **Shuffle**:
  - ↪ assigns keys (words) to each reducer,
  - ↪ sends (k,v) pairs to appropriate reducer
- **Reducer**
  - ↪ just has to sum up the ones

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

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

Hands-on 4: Playing with Hadoop

Your Turn!

- Now you are ready to play with the installed Hadoop

Hands-on 4

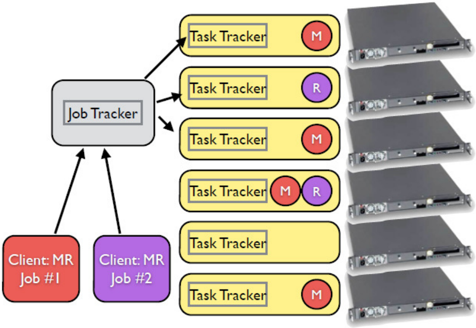
http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/hadoop/wordcount

- Test the `tools/Hadoop` modules in Single mode — Step 1
  - ↪ setup the wordcount example
- Enable a Cluster Setup — Step 2

Summary

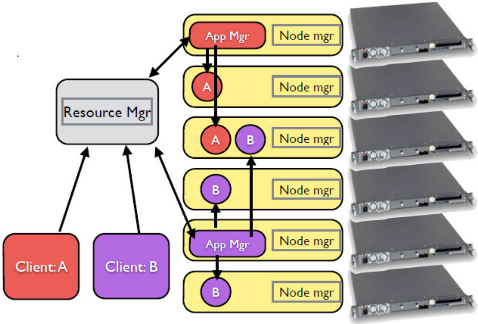
Hadoop 0.1x

- 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 and Hadoop 2

- **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)**.

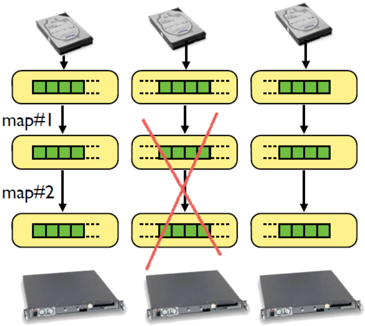
Apache Spark

- 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)

RDD

- Resilient Distributed Dataset (RDD)
  - ↪ Partitioned collections (lists, maps..) across nodes
  - ↪ Set of well-defined operations (incl map, reduce) defined on these RDDs.

RDD



- Fault tolerance works three ways:
  - ↪ Storing, reconstructing lineage
  - ↪ Replication (optional)
  - ↪ Persistance 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

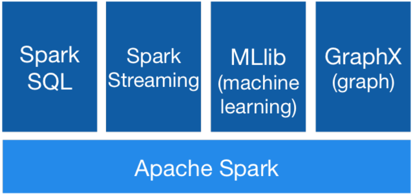
RDD Lineage

- 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

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/spark/install/

- Use EasyBuild to search for a ReciPY for Apache Spark
- Install it and check the installed software

Hands-on 6: Spark Usage

Your Turn!

Hands-on 6

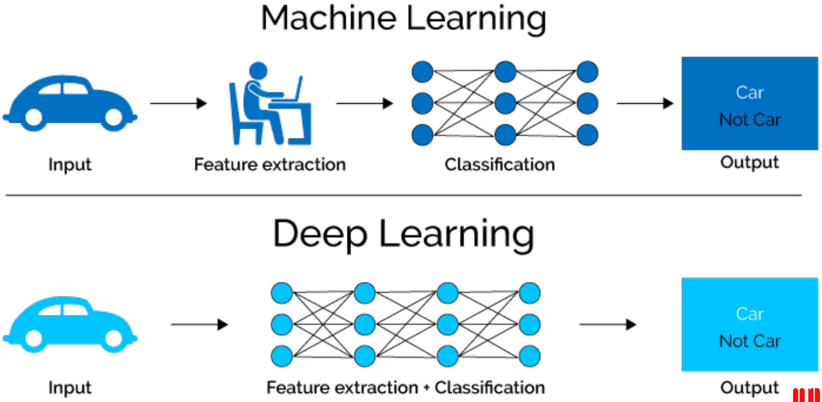
http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/spark/usage/

- Check a single interactive run — Step 1
  - ↪ PySpark, the Spark Python API — Step 1.a.
  - ↪ Scala Spark Shell — Step 1.b.
  - ↪ R Spark Shell **will not be reviewed** here.
- Running Spark standalone cluster — Step 2
  - ↪ In particular, illustrated on Pi estimation.

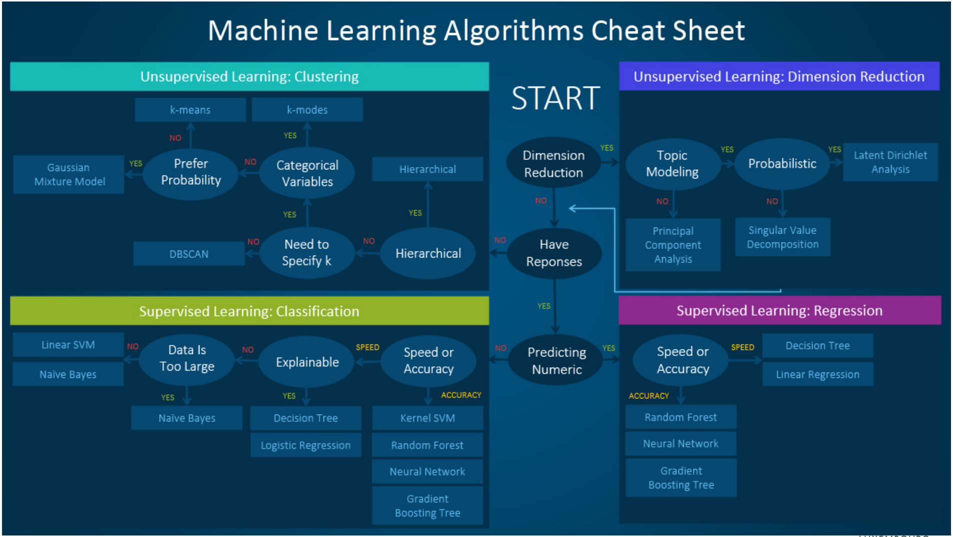
Summary

Big data and Machine/Deep Learning

- **Out-of-scope of this tutorial**:
  ↪ Machine Learning (ML) / Deep Learning theoretical basis



Machine Learning Cheat sheet



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.
- Caffee
  - ↪ a well-known and widely used machine-vision library that ported Matlabs implementation of fast convolutional nets to C and C++
  - ↪ YouâĂŹll also have to consider its successor, Caffee 2,

Machine/Deep-Learning Frameworks

- 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 pythonwrapper
  - ↪ 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.

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

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

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/tensorflow/install/

- Preparation of a Python sand-boxed environment — Step 1
  - ↪ using pyenv and virtualenv
- Tensorflow installation using `pip` — Step 2
- Installation of jupyter Jupyter Notebook — Step 3

Hands-on 8: Playing with Tensorflow

Your Turn!

Hands-on 8

http://nesusws-tutorials-BD-DL.rtfd.io/en/latest/hands-on/tensorflow/mnist/

- Run a very simple MNIST classifier — Step 1
  ↪ MNIST: computer vision dataset (images of handwritten digits)
- Run a deep MNIST classifier using convolutional layers — Step 2
  ↪ compare results with best models

Questions?

http://hpc.uni.lu

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