Hadoop

a quick walk-through

Created by **Foudil BRÉTEL** / cc.in2p3.fr

Resources

Tutorials

- **Yahoo! tutorial** — **2007** *outdated*
- ***Hadoop: The Definitive Guide, 3rd Edition***, O'Reilly 

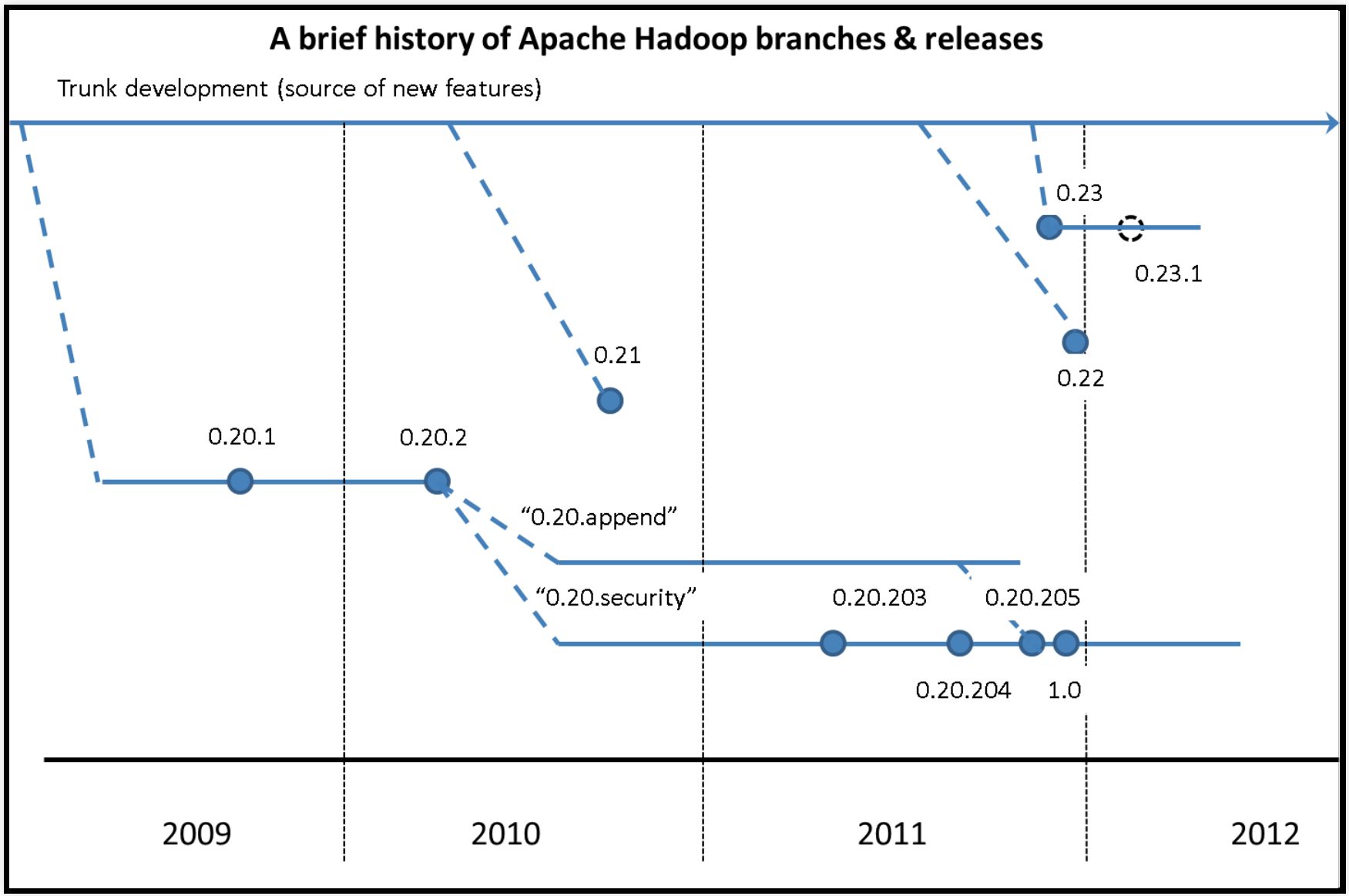
Other resources

- ***Hadoop Operations***, O'Reilly
- **Apache Hadoop documention** (getting started, cluster setup, ...)
- **Cloudera demo VM**, **Cloudera manager**

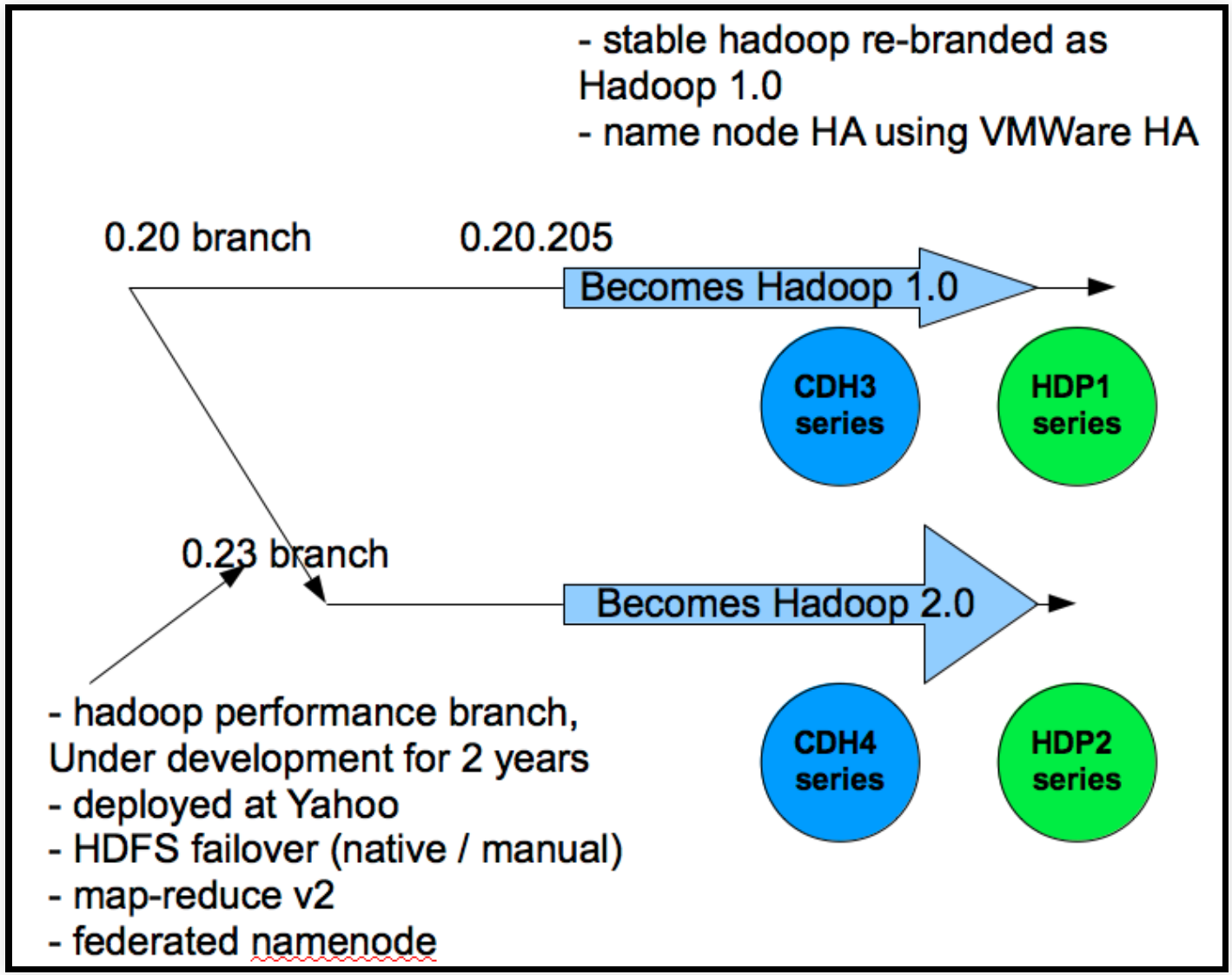
- ***Web Data Management***, Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart

Versions

2005, **Doug Cutting** (Lucene) travaille sur **Nutch**



Versions


- stable hadoop re-branded as Hadoop 1.0
- name node HA using VMWare HA
0.20 branch
0.20.205
Becomes Hadoop 1.0
CDH3 series
HDP1 series
0.23 branch
Becomes Hadoop 2.0
CDH4 series
HDP2 series
- hadoop performance branch, Under development for 2 years
- deployed at Yahoo
- HDFS failover (native / manual)
- map-reduce v2
- federated namenode

Characteristics

*" Hadoop is a large-scale, **distributed**, batch processing infrastructure "*

⇒ parallelism, horizontal-scale

1. simplified programming model
2. distribution of work *and data* across machines (*"data locality"*)

" Grid scheduling of computers can be done with existing systems such as Condor. But Condor does not automatically distribute data: a separate SAN must be managed in addition to the compute cluster. Furthermore, collaboration between multiple compute nodes must be managed with a communication system such as MPI. This programming model is challenging to work with and can lead to the introduction of subtle errors. "

Vocabulary

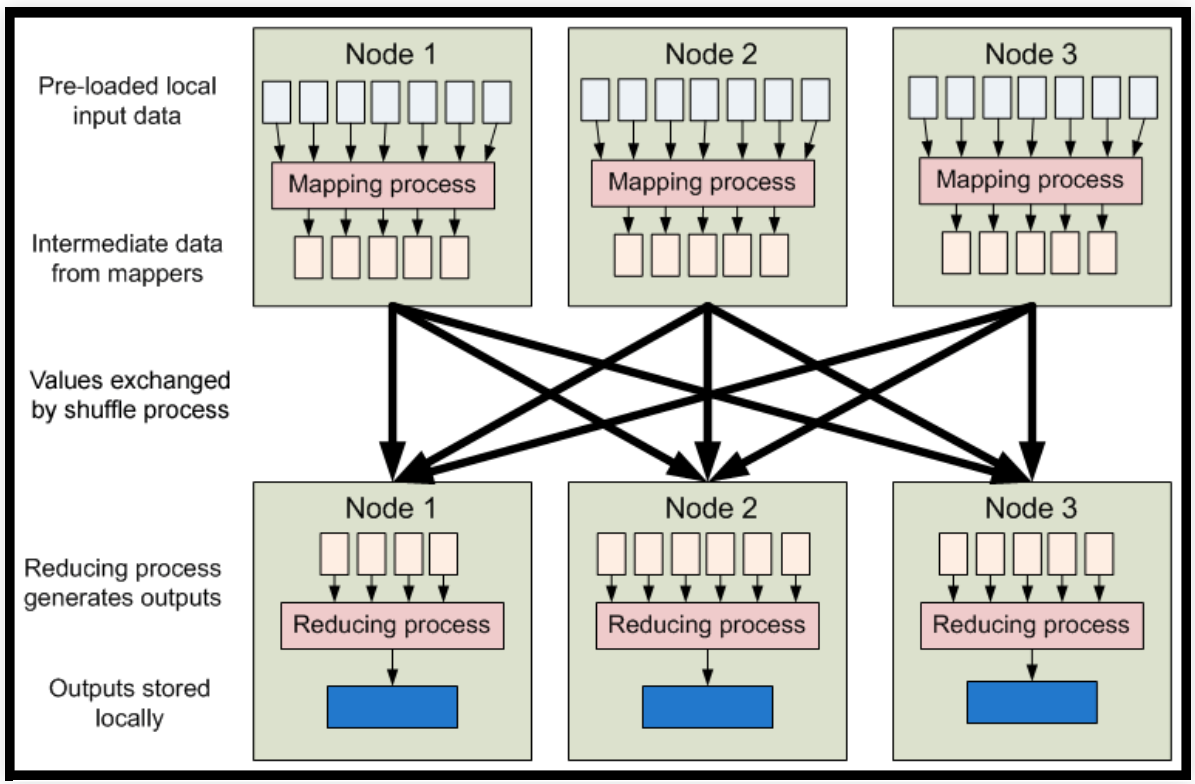
vertical scaling *"scale up"*	add more power (CPU, RAM) to an existing machine
horizontal scaling *"scale out"*	add more machines *

MapReduce Design

Jobs ⊃ Tasks

Tasks are run *in isolation*

- limited *implicit* communication
- reliability (e.g. node failures), **speculative execution**



Vocabulary

flat scalability	write program once, scale out *

HDFS Design

- data is distributed to all the nodes
- large data files are split into *chunks*
- chunks are *replicated* * across several machines (failure-resistent)
- *single namespace* (unix-like)

*** backups for free**

Data is conceptually **record-oriented**

HDFS Considerations

Pros

very large amount of information, reliability, fast access, availability (many clients), …

Cons

- long sequential streaming reads (*no random access*)
- write once, read several times
- …

HDFS

chunks = blocks of a fixed size (64MB *)

chunks are stored on **DataNodes**

The catalog of *chunks* is a *single* service called **NameNode**

DataNodes store data **on top of the OS** file system (not raw devices) **

* 64MB ≫ 4KB (ext4, NTFS) ⇒ list of blocks per file smaller

** FS tuning welcome (ext4 options + mount options, see **Hadoop Operations, O'Reilly**)

Commandes

- `hadoop`
- `hdfs`
- `mapred`

MapReduce Exercices

```
0057
332130  # USAF weather station identifier
99999   # WBAN weather station identifier
19500101# observation date
0300    # observation time
4
+51317  # latitude (degrees x 1000)
+028783 # longitude (degrees x 1000)
FM-12
+0171   # elevation (meters)
99999
V020
320     # wind direction (degrees)
1       # quality code
N
0072
1
00450   # sky ceiling height (meters)
1       # quality code
C
N
010000  # visibility distance (meters)
1       # quality code
N
9
-0128   # air temperature (degrees Celsius x 10)
1       # quality code
-0139   # dew point temperature (degrees Celsius x 10)
1       # quality code
10268   # atmospheric pressure (hectopascals x 10)
1       # quality code
```

Use cases

Last.fm

50 nodes, 300 cores, 100 TB disk

Usages

logfile analysis, evaluation of A/B tests, ad hoc processing, charts generation

Adoption motivations

- distributed filesystem = *redundant backups* (web logs, user listening data, …) at no extra cost.
- cheap commodity hardware ⇒ *scalability*
- flexible, easy distributed computing
- open source advantages (no cost, customizable)

Facebook

worldwide second-largest Hadoop cluster

2+ PB disk (+10TB/day), 2,400 cores, ~9 TB RAM

Usages

- *daily and hourly* reports/analyses about growth of the users, page views, average time spent on the site, advertisement performance, …
- backend processing for site features (ex: suggestions)
- *de facto long-term archival store for logs*

Facebook (continued)

Adoption motivations

Before

- initial data warehousing didn't scale (entirely on an Oracle® instance)
- favourable preconception (Yahoo! using it, Google MapReduce)

After prototype

- ability to use your favorite programming language (Hadoop streaming)
- datasets published *in one centralized data store*
- customizable ⇒ Hive

Rackspace

18 nodes, 22+ TB

Usage

scale at aggregating and indexing (Lucene) Postfix and Exchange logs

Adoption motivations

log processing previously based on MySQL, but…
RDBMS sharding ⇒ lose advantages of SQL

HBase

Google's *BigTable* clone: distributed, versioned, **column-oriented** on top of HDFS

Provides fast record lookups/updates for large tables (hundreds of millions+ rows)

Built with **very large scale** and **distribution** in mind.

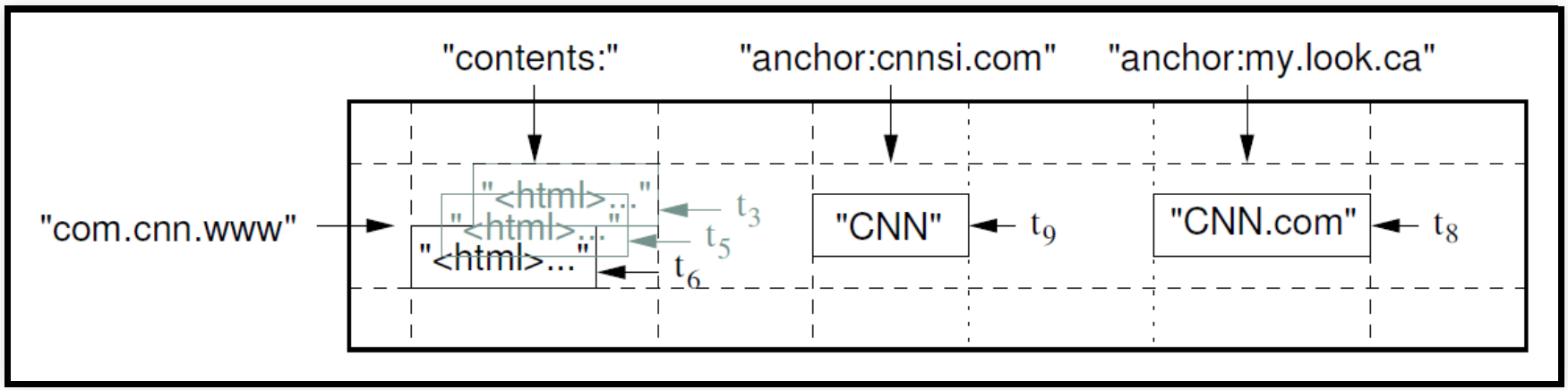
Production users include Facebook (messaging system) Adobe, StumbleUpon, Twitter (people search), and groups at Yahoo!

> *"A Bigtable is a sparse, distributed, persistent multidimensional sorted map."*

Column-*family*-oriented

physically, data saved *by column* ***families***

conceptualy, data organized in tables



- **row/key** = **entry point**, byte array, unique, lexicographically sorted
- **column family** = rigid* columns stored in same **HFile**, share same options (e.g. compression)
- **column (members)** = versioned (**timestamp**) members
- cell = {row, column_family:column (, version)}. empty cells are not stored
- regions ← automatic sharding

* must be define at table creation, cannot be added

multi-dimensional

```
{
  "1" : {
    "A" : "x",
    "B" : "z"
  },
  "aaaaa" : {
    "A" : "y",
    "B" : "w"
  },
  "xyz" : {
    "A" : "hello",
    "B" : "there"
  }
}
```

multi-dimensional (continued)

```
{
  // ...
  "aaaaa" : {
    "A" : {
      "foo" : "y",
      "bar" : "d"
    },
    "B" : {
      "" : "w"
    }
  },
  "aaaab" : {
    "A" : {
      "foo" : "world",
      "bar" : "domination"
    },
    "B" : {
```

Row `aaaaa` has 3 rows: `A:foo`, `A:bar` and `B:`

To know all columns in all rows ⇒ full table scan

THE END

BY Foudil BRÉTEL / cc.in2p3.fr