Apache Hadoop Ecosystem

Rim Moussa

ZENITH Team Inria Sophia Antipolis
***DataScale* project**
`rim.moussa@inria.fr`

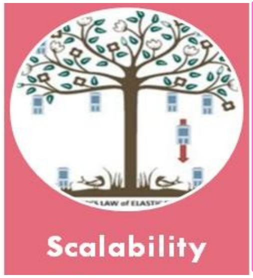
26th, Feb. 2015

ENSMA Poitiers Seminar Days

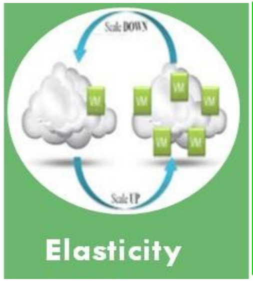
Context *large scale systems

- Response time (RIUD ops: one hit, OLTP)
- Processing Time (analytics: data mining, OLAP workloads)

- Continuity of service despite nodes' failures
  - Data recovery
  - Query/Job recovery



- System performance face to *n times* higher loads + *n times* hardware capacities



- Automatic provisioning and relinquish of resources
- Storage: bucket split/merge

- Cost in-premises
- Cost at a CSP

Context *categorization

Classical	Columnar	MapReduce	Dataflow	Array DB
				Graph DB
				...

Microsoft SQL Server

IBM DB2

GREENPLUM

monetdb

vectorwise

APACHE HBASE

hadoop

HIVE

Spark

Google Cloud Platform

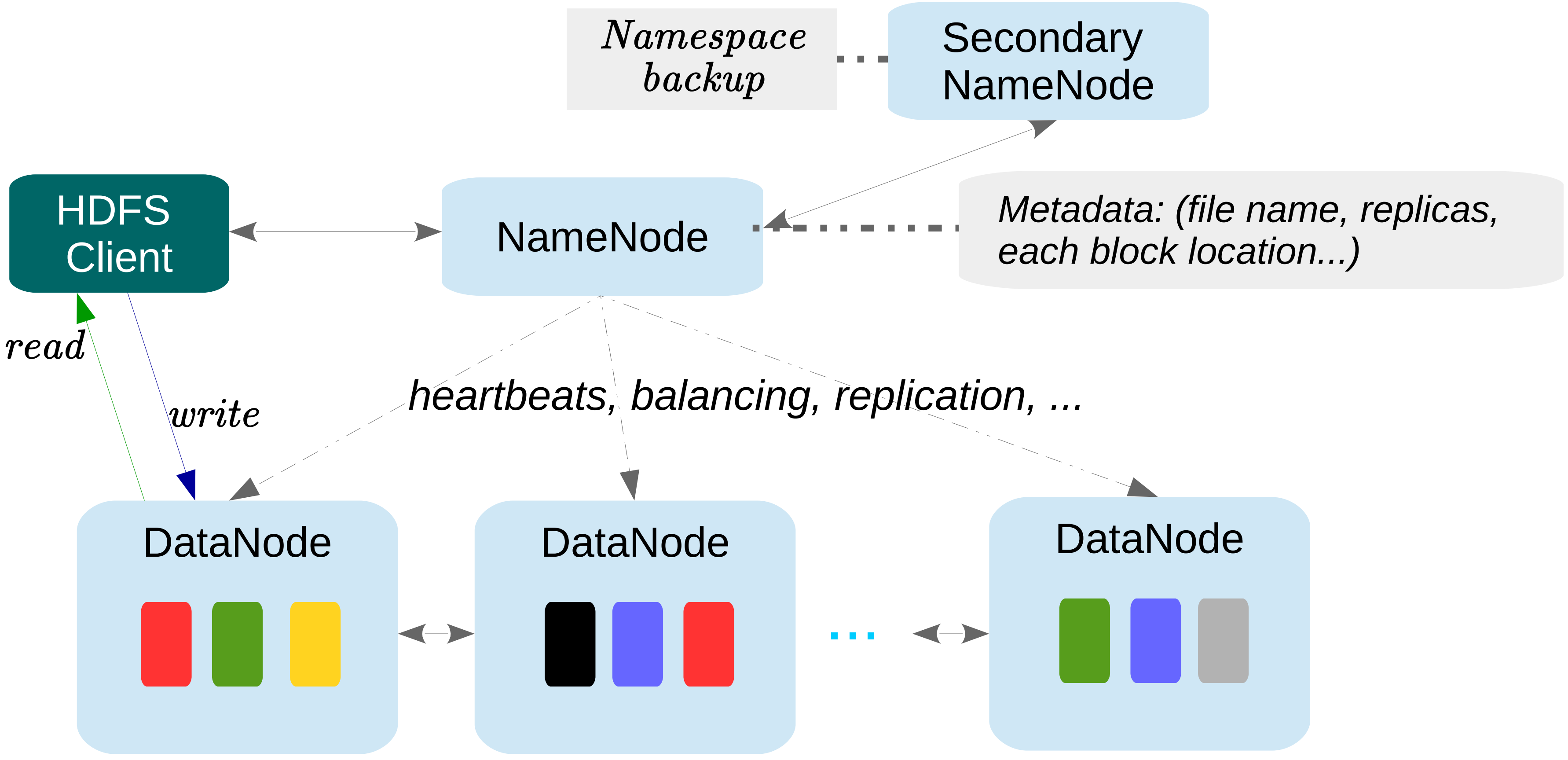
Apache Hadoop Ecosystem

- **HDFS: Distributed File System**
- **MapReduce**: parallel data processing
- **Pig latin**: data flow scripting language
- **HBase**: distributed, columnar, non-relational database
- **Hive**: data warehouse infrastructure + HQL
- **ZooKeeper:** centralized service providing distributed synchronization
- **Ganglia**: monitoring system for clusters and grids
- **Sqoop**: tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores (RDBMS)
- **Hama**: distributed engine for massive scientific computations such as matrix, graph and network algorithm (BSP)
- **HCatalog**: table mgmt layer for Hive metadata to other Hadoop applications
- **Mahout**: scalable machine learning library.
- **Ambari**: software for provisioning, managing, and monitoring Apache Hadoop clusters
- **Flume**: distributed service for efficiently collecting, aggregating, and moving large amounts of log data
- **Giraph**: iterative graph processing system
- **DRILL**: low latency SQL query engine for Hadoop
- **Oozie or TEZ**: workflow automation

Hadoop Distributed File System (HDFS)

- Distributed File Systems
  - Network File System (Sun Microsystems, 1984), ...
  - Google File System (Google, 2000)
- Large scale distributed data intensive systems
  - big data, I/O-bound applications
- Key properties
  - High-throughput
    - Large blocks: 256MB,.. versus common kilobyte range blocks (8KB, ..)
  - Scalability
    - Yahoo requirements for HDFS in 2006 were,
      - storage capacity: 10 PB,
      - number of nodes: 10,000 (1TB each),
      - number of concurrent clients: 100,000, ...
    - K. V. Shvachko. *HDFS Scalability: the limits to growth*.
      - Namespace server RAM correlates to with the storage capacity of hadoop clusters.
  - High availability
    - Achieved through blocks' replication

Hadoop Distributed File System



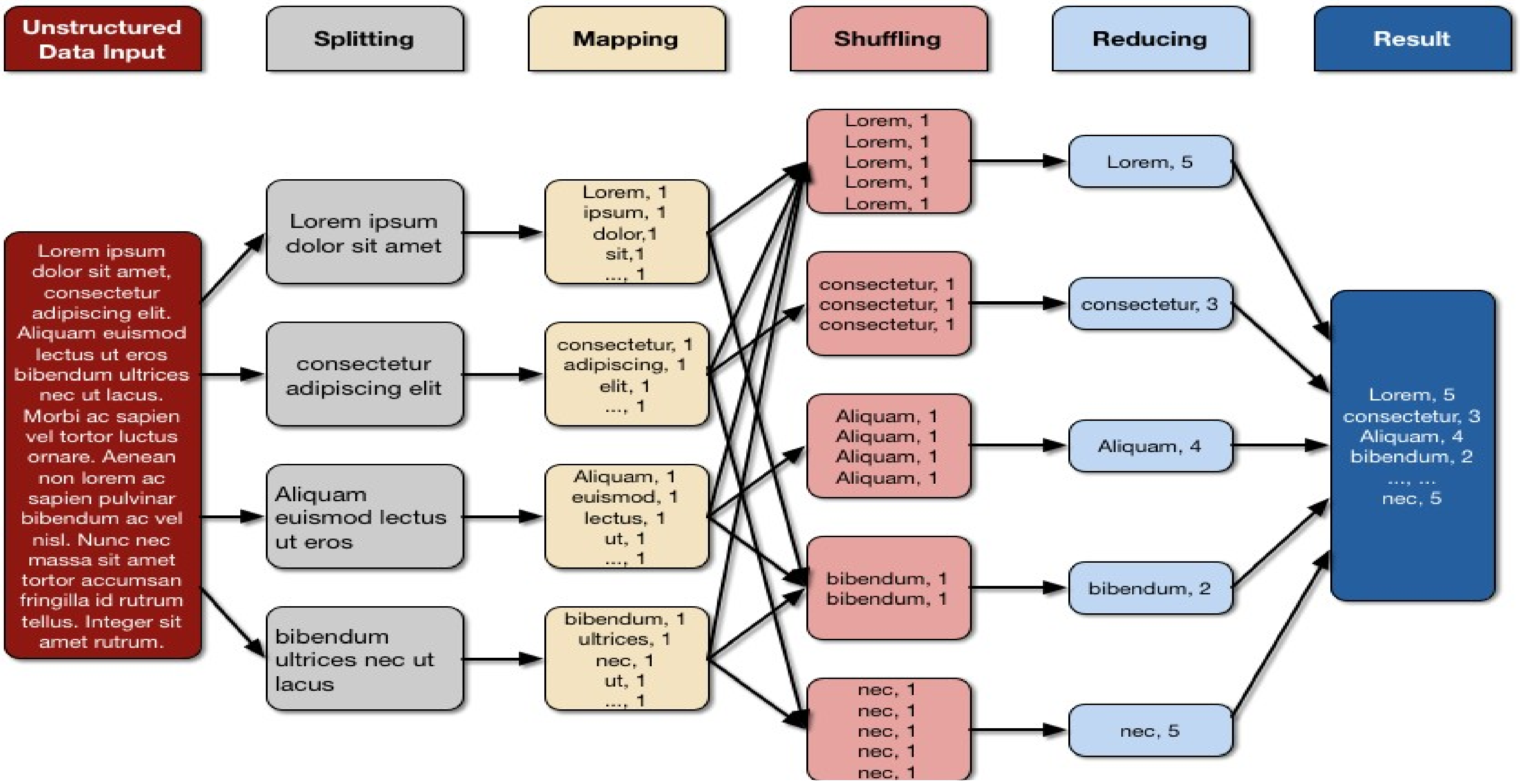
» *HDFS client* asks the *Name Node* for metadata, and performs reads/writes of files on *DataNodes*.

» *Data Nodes* communicate with each other for pipeline file reads and writes.

MapReduce Framework

- Google MapReduce (by J. Dean and S. Ghemawat, 2004)
- A framework for large scale parallel computations,
  - Users specify computations in terms of a *Map* and *Reduce* function. The system automatically parallelizes the computation across large-scale clusters.
  - `Map (key, value) --> list(key', value')`
    - Mappers perform the same processing on partitioned data
  - `Reduce (key', list(value')) --> list(key', value”)`
    - Reducers aggregate the data processed by Mappers
- Key properties
  - Reliability achieved through job resubmission
  - Scalability
    - Cluster hardware
    - Data volume
    - Job complexity and patterns
      - Adequacy of the framework to the problem

Distributed Word Count Example



Excerpt of MR Word Count code

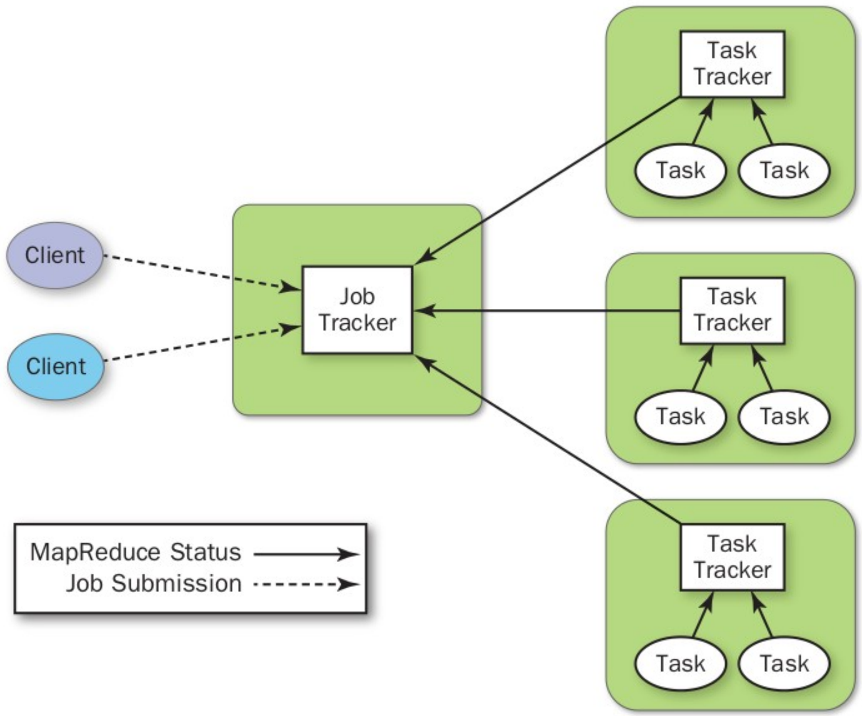
```java
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
}

public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

--Word Count Example (ctnd 1)

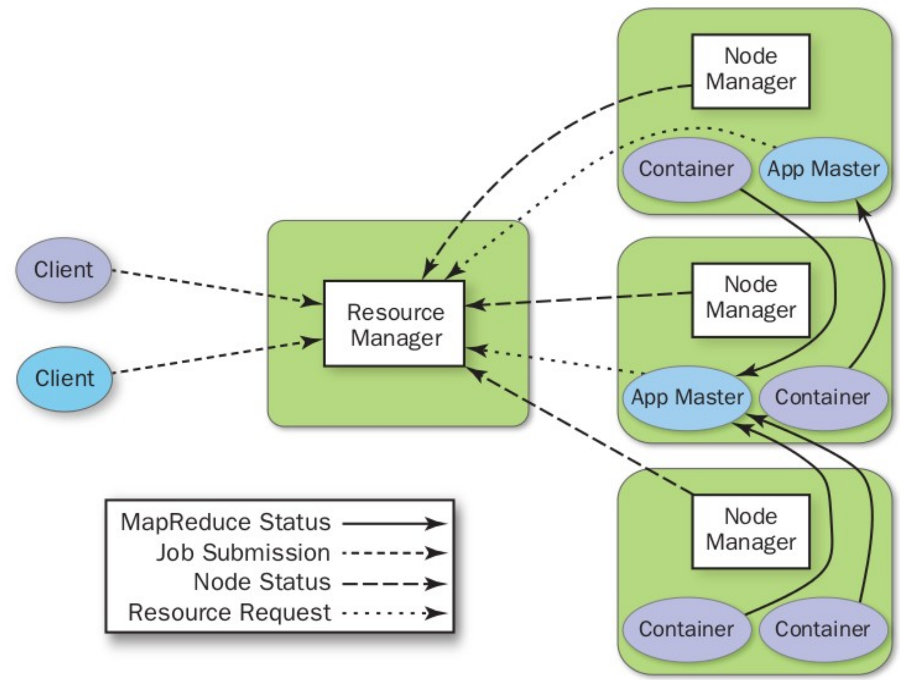
```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```

Hadoop 0|1.x versus Hadoop YARN

Hadoop 0|1.x

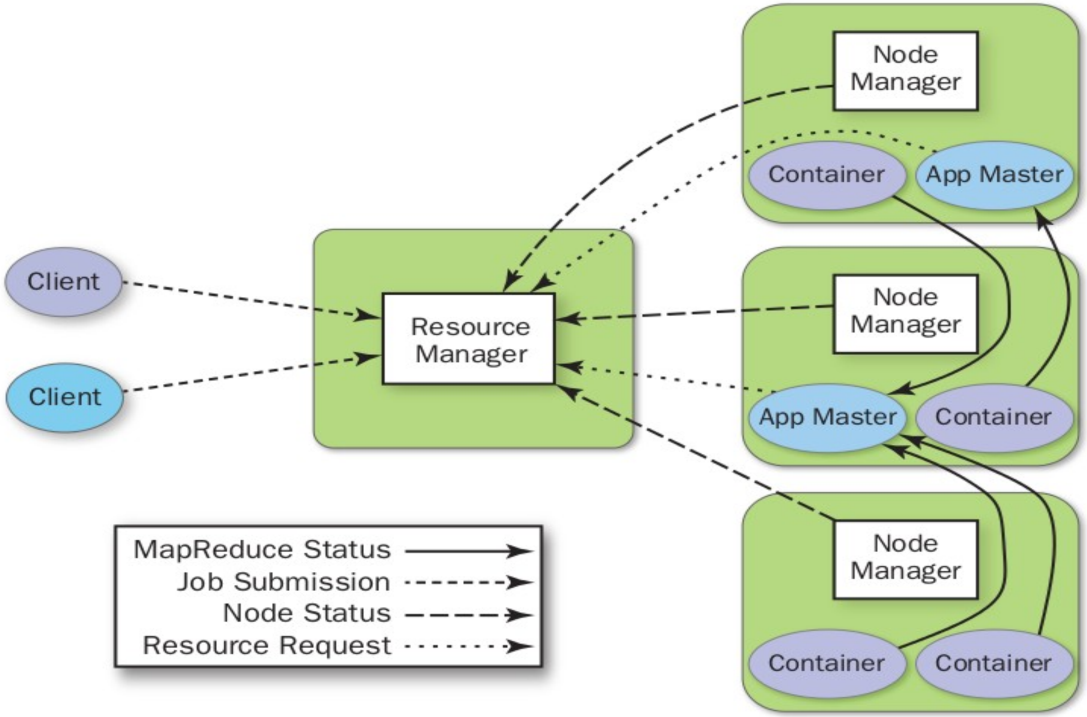


Hadoop YARN



- Static resource allocation deficiencies
- *Job Tracker* manages cluster resources and monitors MR Jobs

Hadoop YARN * Job processing



- *Application Master* manages the application's lifecycle, negotiates resources from the *Resource Manager*
- *Node Manager* manages processes on the node
- *Resource Manager* is responsible for allocating resources to running applications,
- *Container* (YARN Child) performs MR tasks and has its CPU, RAM attributes

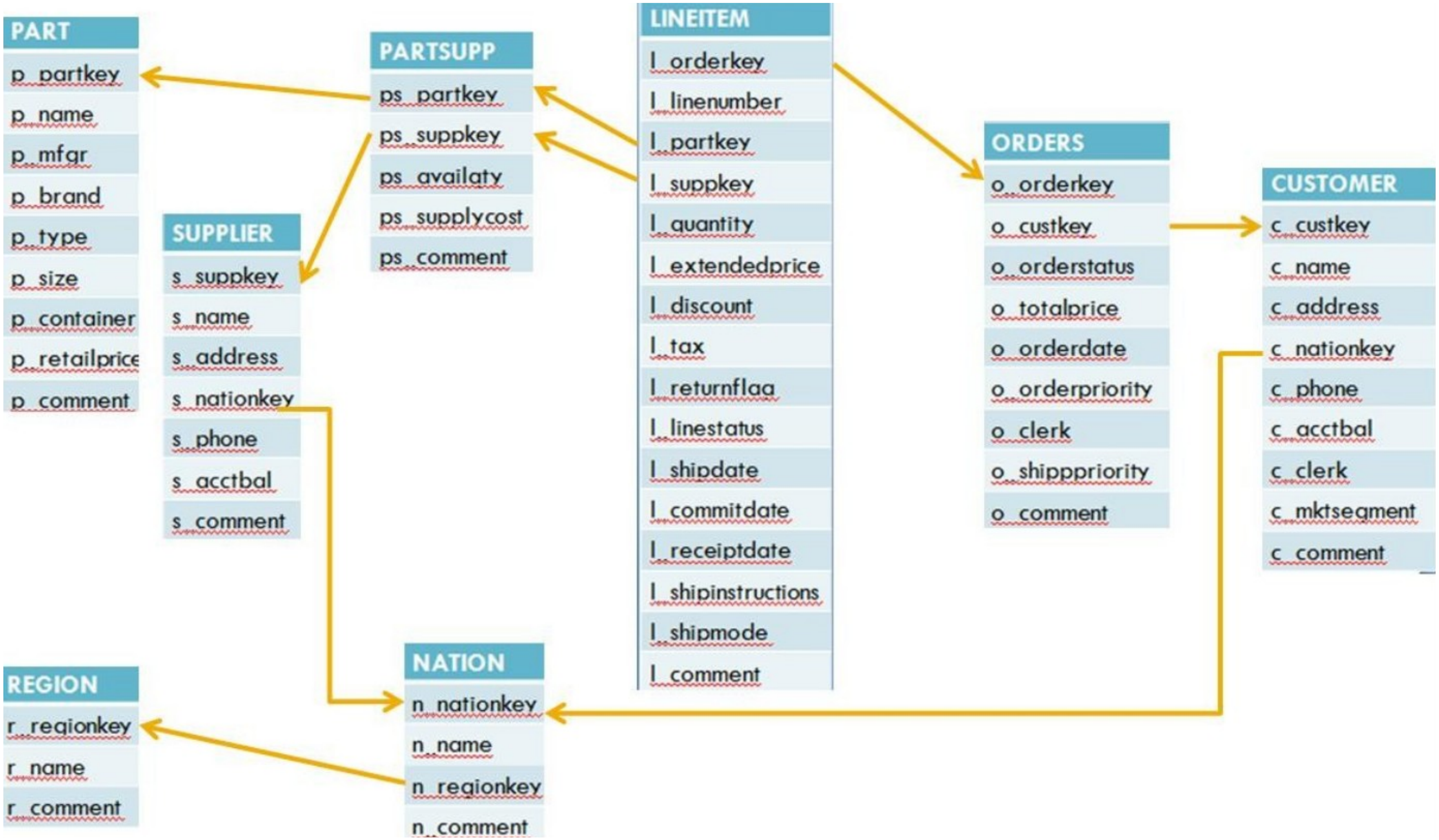
MR Jobs Performance Tuning

- I/O
  - Data Block Size
    - Can be set for each file
- Parallelism
  - Input Split --> Number of mappers
  - Number of Reducers
- Data Compression during shuffle
- Resource Management
  - Each Node has different computing and memory capacities
  - Mapper & Reducer allocated resources
    - might be different in Hadoop YARN
- Code
    - Implement combiners (local reducers)
    - lower data transfer cost

Pig Latin

- Google Sawzall (R. Pike et al. 2005)
- High-level parallel data flow language
  - Open-source MapReduce Code
    - Basic operators: boolean ops, arithmetic ops, cast ops, ...
    - Relational operators: filtering, projection, join, group, sort, cross, ..
    - Aggregation functions: avg, max,count, sum, ..
    - Load/Store functions
    - *Piggybank.jar*: open source of UDFs
- Apache Oozie then Apache Tez
  - Open-source workflow/coordination service to manage data processing jobs for Apache Hadoop
  - A Pig script is translated into a series of MapReduce Jobs which form a DAG (Directed Acyclic Graph)
    - A data flow (data move) is an edge
    - Each application logic is a vertice

Pig Example *TPC-H relational schema



Pig Example *Q16 of TPC-H Benchmark

<Q16> The *Parts/Supplier Relationship Query* counts the number of suppliers who can supply parts that satisfy a particular customer's requirements. The customer is interested in parts of eight different sizes as long as they are not of a given type, not of a given brand, and not from a supplier who has had complaints registered at the Better Business Bureau.

```
SELECT p_brand, p_type, p_size, count(distinct ps_suppkey)
as supplier_cnt
FROM partsupp, part
WHERE  p_partkey = ps_partkey
AND p_brand <> '[BRAND]'
AND p_type NOT LIKE '[TYPE]%'
AND p_size in ([SIZE1], [SIZE2], [SIZE3], [SIZE4], [SIZE5],
[SIZE6], [SIZE7], [SIZE8])
AND  ps_suppkey NOT IN (SELECT s_suppkey FROM   supplier
                    WHERE s_comment like '%Customer
%Complaints%')
GROUP BY p_brand, p_type, p_size
ORDER BY  supplier_cnt DESC, p_brand, p_type, p_size;
```

Pig Example *Q16 of TPC-H Benchmark

```
--- Suppliers with no complaints
supplier = LOAD 'TPCH/supplier.tbl' USING PigStorage('|') AS
(s_suppkey:int, s_name:chararray, s_address:chararray, s_nationkey:int,
s_phone:chararray, s_acctbal:double, s_comment:chararray);
supplier_pb= FILTER supplier BY NOT(s_comment matches
'.*Customer.*Complaints.*');
suppkeys_pb = FOREACH supplier_pb GENERATE s_suppkey;
--- Parts size in 49, 14, 23, 45, 19, 3, 36, 9
part = LOAD 'TPCH/part.tbl' USING PigStorage('|') AS (...);
parts = FILTER part BY (p_brand != 'Brand#45') AND  NOT (p_type matches
'MEDIUM POLISHED.*') AND (p_size IN (49, 14, 23, 45, 19, 3, 36 , 9);
---Join partsupp, selected parts, selected suppliers
partsupp = LOAD 'TPCH/partsupp.tbl' using PigStorage('|') AS (...);
part_partsupp = JOIN partsupp BY ps_partkey, parts BY p_partkey;
not_pb_supp = JOIN part_partsupp BY ps_suppkey, suppkeys_pb BY s_suppkey;
selected = FOREACH not_pb_supp GENERATE ps_suppkey, p_brand, p_type,
p_size;
grouped = GROUP selected BY (p_brand,p_type,p_size);
count_supp = FOREEACH grouped GENERATE flatten(group),
COUNT(selected.ps_suppkey) as supplier_cnt;
result = ORDER count_supp BY supplier_cnt DESC, p_brand, p_type, p_size;
STORE result INTO 'OUTPUT_PATH/tpch_query16';
```

DataScale @ZENITH,Inria Sophia Antipolis

- With Florent Masseglia, Reza Akhbarinia and Patrick Valduriez
- Partners
  - Bull (ATOS), CEA, ActiveEon, Armadillo, linkfluence, IPGP
- DataScale
  - Applications which develop Big Data technological building blocks that will enrich the HPC ecosystem,
    - Three specific use cases :
      - Seismic event detection
      - Management of large HPC Cluster
      - Multimedia product analysis.
- ZENITH *Inria
  - Use case *Management of large HPC Cluster*
    - Large-scale and Scalable Log Mining
      - Implementation of state-of-the-art algorithms,
      - Proposal & Implementation of new algorithms
      - Implementation of a Synthetic Benchmark
      - Tests with real datasets provided by our partners
      - Deployment at Bull

Conclusion
Extensions?

- HDFS
  - *Quantcast File System*: uses erasure codes rather than replication for fault tolerance
  - *Spark*: Resilient Distributed Dataset --> in-memory data storage
- Data Mining
  - MapReduce for Iterative jobs?
  - Projects addressing Iterative Jobs for Hadoop 1.x: Peregrine, HaLoop, ..
- OLAP
  - Join operations are very expensive
    - CoHadoop implements *Data Colocation* --> Most efficient way to perform large scale joins in parallel
  - Indexes:
    - Hadoop++ (20×), HAIL (65×faster) than Hadoop: by J. Dittrich et al.
    - Indexing data is stored in HDFS blocks read by Mappers
  - The *Workflow manager* orchestrates jobs, and should implement advanced optimization techniques (metadata about data flows, //DAG)

Conclusion
What beyond MapReduce?

- Apache Hadoop YARN
  - Hadoop YARN Is open to integrate new frameworks for parallel data processing: YARN Applications!
- Other Computational Models
  - BSP (Bulk Synchronous Parallelism): Apache Hama, Pregel (Apache Giraph)
  - Multi-level serving trees, Dremel (Google BigQuery), Apache Drill, Cloudera Impala

Thank you for Your Attention

Q & A

Apache Hadoop Ecosystem

ENSMA Poitiers Seminar Days

26th Feb., 2015