Improving the MapReduce Big Data Processing Framework

Miguel Liroz Gistau, Reza Akbarinia, Patrick Valduriez
INRIA & LIRMM, Montpellier, France

In collaboration with
Divyakant Agrawal, UCSB
Esther Pacitti, UM2, LIRMM & INRIA

4th Wrokshop of Project HOSCAR
Contents

MapReduce Overview

Contributions

• MRPart: reducing data transfers in shuffle phase
• FP-Hadoop: making reduce phase more parallel
• hadoop_g5k: repeatable tests in Grid5000 platform

Conclusions

• Beyond MapReduce
MAPREDUCE OVERVIEW
MapReduce Overview

Programming model and framework
- Developed by Google for big data parallel processing in data centers
  - e.g., PageRank algorithm, inverted indexes
- Used in combination with other Google services (GFS, BigTable,…)
- Hadoop: an open-source implementation of MapReduce

Design requirements
- Executed on commodity clusters
- Failures are the norm rather than the exception

Goal
- Automatic parallelization and distribution and fault tolerance

Principle
- Data locality: move computation to data
Programming model

Data consists of key-value pairs (tuples)

Functions

• map: \((k_1,v_1) \rightarrow \text{list}(k_2,v_2)\)
  – Processes input key-value pairs
  – For each input pair produces a set of intermediate pairs

• reduce: \((k_2,\text{list}(v_2)) \rightarrow \text{list}(k_3,v_3)\)
  – Receives all the values for a given intermediate key
  – For each intermediate key produces a set of output pairs
MapReduce Example

**Wordcount**: count the frequency of each word in a big file

map(key, value)
// key: offset, value: a line
for each word w
    emit(w,1)

reduce(key, values)
// key: a word, values: list of counts
count = 0
for each v in values
    count += v
emit(key, count)
MapReduce Job Execution

![MapReduce Job Execution Diagram]

Intermediate Keys (IKs)

Improving the MapReduce Big Data Processing Framework

Miguel Liroz
Architectural view

Client

Submit job

Master

HDFS

Worker

Local disk

chunk_0

chunk_1

chunk_2

file_in_1

file_in_2
Architectural view

Client

Master

Create splits

HDFS

Input

file_in_1

chunk_0

split_0

file_in_2

chunk_0

split_2

Worker

Local disk

Worker

Local disk
Architectural view
Improving the MapReduce Big Data Processing Framework

Miguel Liroz
Improving the MapReduce Big Data Processing Framework

Architectural view

Client

Master

Schedule reduce tasks

HDFS

Input

Worker

m₀

Local disk

split₀

chunk₀

file_in₁

Worker

m₁

Local disk

split₁

chunk₁

file_in₂

Worker

m₂

Local disk

split₂

chunk₂

file_in₃
Architectural view

Client

Master

Schedule reduce tasks

HDFS

Input

file_in_1

chunk_0

split_0

Worker

m_0

Local disk

file_in_2

chunk_0

split_2

Worker

m_1

Local disk

Worker

m_2

Local disk

Worker

r_0

Worker

r_1
Architectural view

Improving the MapReduce Big Data Processing Framework

Miguel Liroz
Improving the MapReduce Big Data Processing Framework

Miguel Liroz
Shuffle Phase

Partitioning, sorting and transfer of data between map and reduce

Steps

• In the map task
  – Intermediate pairs are partitioned into R fragments
    – By default $part(key) = hash(key) \mod |R|$
  – Pairs are sorted by key within each partition

• In the reduce task
  – Pairs with the same key are merged into a single $(k_2, list(v_2))$ pair and sent to the reduce function
Fault-Tolerance

Failures are the norm rather than the exception in large-scale data centers

Failure of workers
- Periodic heartbeat messages to the master
  - Finished map task and map and reduce tasks in progress are rescheduled

Failure of the master
- Periodic checkpoints to the DFS

Slow workers (stragglers)
- When all tasks are scheduled, running task are speculatively rescheduled in idle workers
OUR CONTRIBUTIONS
Overview

Shuffle overhead

• MRPart: minimizing data transfers between mappers and reducers

Skew prevention

• FP-Hadoop: parallelization of reduce phase with a multi-iteration intermediate phase

Experiment workflow

• hadoop_g5k: available tool for repeatable tests in Grid5000 platform
1. MR-Part: Improving Reduce Locality

Motivation

• The shuffle phase may involve big data transfers
• During shuffle, nodes are competing for bandwidth
• Result: some jobs are slowed down while this phase is completed

Ideal case

• No data transfer
  – All values for an intermediate key are produced in the same worker
  – They are assigned to a reduce task executed by the same worker
Motivation

Normal situation

Worker 1

Map₀

Reduce₀

Worker 2

Map₁

Reduce₁

Map₂

Map₃

Ideal case

Worker 1

Map₀

Reduce₀

Worker 2

Map₁

Reduce₁

Map₂

Map₃

Colors represent tuples producing same IK (Intermediate Key)
Main Idea of MR-Part

Given a file F and a set of MR jobs → Goal: minimize shuffle data transfer

Partitioning input data

- Tuples generating the same IK are placed together
- Rationale: they all go to the same reducer
Main Idea of MR-Part

Given a file F and a set of MR jobs → Goal: minimize shuffle data transfer

Partitioning input data

- Tuples generating the same IK are placed together
- Rationale: they all go to the same reducer
Improving the MapReduce Big Data Processing Framework

**MR-Part Approach**

- **Monitoring**
  - Monitoring job execution
  - Metadata

- **Repartitioning**
  - Workload modeling
  - Graph partitioning
  - Input repartitioning

- **Execution & Scheduling**
  - Locality aware-scheduling
Experiments

Environment
- **Grid5000**

Comparison
- Native Hadoop (NAT)
- Hadoop + reduce locality-aware scheduling (RLS)
- MR-Part (MRP)

Benchmark
- TPC-H, MapReduce version

Parameters
- Data size, cluster size, bandwidth

Metrics
- Transferred data
- Latency (response time)
Percentage of Transferred Data

Different type of queries

Varying cluster and data size
Varying bandwidth

TPC-H Q17

TPC-H Q9
2. FP-Hadoop: Making Reduce Phase More Parallel

Parallelization in map phase

- Input data is divided into splits of similar size
- Map tasks are scheduled in free workers
  - Each map task consumes one of the splits

Parallelization in the reduce phase

- Intermediate keys are assigned to reduce tasks depending on a function
- Size of reduce tasks cannot be defined a priori
  - Even with ideal partitioning function, keys with a lot of values still produce overloaded splits
2. FP-Hadoop: Making Reduce Phase More Parallel
Main Idea of FP-Hadoop

Reduce input data is divided into splits (IR splits)

- The size of the splits is bounded
- Splits are consumed in the same way as in the map phase

Reduce function is divided into two functions

- Intermediate reduce: parts that can be done in parallel
  - Eventually it can be executed in multiple phases
- Final reduce: performs the final grouping
Main Idea of FP-Hadoop

Input Splits → Map workers → Intermediate key-values → Intermediate reduce workers → Intermediate key-values → Final reduce workers → Results

\[ D_1 \rightarrow ... \rightarrow D_n \rightarrow M_1 \rightarrow ... \rightarrow M_m \rightarrow k_1 \rightarrow k_2 \rightarrow k_3 \rightarrow R_1 \rightarrow ... \rightarrow R_r \rightarrow k_1' \rightarrow k_2' \rightarrow k_3' \rightarrow R_1' \rightarrow ... \rightarrow R_r' \rightarrow O_1 \rightarrow ... \rightarrow O_r \]
FP-Hadoop approach

A modified scheduler is injected into MR framework

• The scheduler selects a subset of values of each key and creates an IR split
• IR split is assigned to an IR task and allocated in a worker
• This process can be repeated several times
• At the end, a final reducer regroups the values of each key
Experiments

Environment

- Grid5000

Comparison

- Native Hadoop (NAT)
- FP-Hadoop (FPH)

Benchmark

- Top-k query (sort, pagerank, inverted index)
- Synthetic data set, Wikipedia data

Parameters

- Data size, cluster size, Skew, FPH conf parameters

Metrics

- Latency (response time)
Results

Comparison with native Hadoop

Synthetic dataset, top-k query

Wikipedia stats, top-k query
Results

Comparison with native Hadoop

![Different queries diagram](image)

![Cluster size diagram](image)
Results

Comparison with native Hadoop

Effect of Data Skew

Response time vs. number of intermediate keys

Skew (zipf exponent)

Number of keys

Improving the MapReduce Big Data Processing Framework

Miguel Liroz
3. hadoop_g5k: Repeatable tests in Grid5000

Experimental evaluation in Grid5000 platform

- French grid infrastructure deployed over 11 sites
- Aims to provide “highly reconfigurable, controllable and monitorable experimental platform to its users”

Hadoop_g5k

- A tool to facilitate repeatable tests with Hadoop in the Grid500 platform
- Publicly available

https://github.com/mliroz/hadoop_g5k
CONCLUSION
Summary

Overview of MapReduce

Contributions

• Proposed prototypes:
  – MR-Part: reducing data transfer in shuffle phase
  – FP-Hadoop: making reduce phase more parallel
• Hadoop_g5k: repeatable experiments in Grid5000
Beyond MapReduce

Great interest of industry in MapReduce

- Amazon Elastic MapReduce, MapR, Cloudera, Hortonworks, IBM BigInsights

Hadoop ecosystem

- HBase, Hive, Pig, YARN, Mahout, Oozie

Post-Hadoop frameworks

- Google Dremel
- Apache Spark
Thank you

Questions?