PER-MARE: Adaptive Deployment of MapReduce over Pervasive Grids

STIC-AmSud 13STIC-07 project

L. A. Steffenel, O. Flauzac, A. Schwertner Charão, P. Barcelos, B. Stein, S. Nesmachnow, M. Kirsch Pinheiro et C. Souveyet

RGE – Réseaux Grand Est
13 Février 2014 - Reims, France
Outline

Motivation
The PER-MARE project
Results from the first year
Discussions & Future work
Conclusions
MapReduce

- MapReduce as a computing paradigm
  - Proposed in 2004 by Google
  - Best known by Hadoop, an Apache project since 2005
- MapReduce basics
  - Map-Reduce algorithm = job
  - Operates with **key-value** pairs: (k,V)
  - MR Job defined by 2 functions
    - map: \((k1;v1) \rightarrow \{(k2;v2)\}\)
    - reduce: \((k2;\{v2\}) \rightarrow \{(k3;v3)\}\)
Pervasive Grids vs Cloud

• Cloud computing and MapReduce
  ✓ Elasticity of resources
  ✗ Sensible data cannot be deployed on the cloud

• Why not use the enterprises own resources to provide computing power?

• Pervasive Grids
  • Computing environment built over available resources
  • "Costless" solution

• Challenges
  • High volatility of resources
  • High heterogeneity
MapReduce Today

• Hadoop is not tailored for heterogeneous and volatile environments
  • Extensive configuration parameters
  • Limited to disconnections – no join
• A few works tried to port MapReduce to dynamic environments but lack some essential points
  • API compatibility
  • Adaptive scheduling
  • High volatility support
The PER-MARE Vision

The PER-MARE project aims to adapt MapReduce to pervasive grids under a two-fold approach

• Improving Hadoop
  • Context-aware task scheduling
  • Automatic tuning of nodes
  • Improved fault tolerance

• Developing a MapReduce P2P framework
  • Compatible with Hadoop API
  • Easy to deploy in pervasive environments
Some Results from this First Year

• Porting MapReduce to a P2P environment
  • Distribute tasks and data in a more efficient way
• Improve fault-tolerance on Hadoop
  • Elimination of single points of failure
  • Design of a testbed for better reproducibility of tests
• Context-aware scheduling of Hadoop jobs
  • Use system probes to describe nodes' capabilities
Porting MapReduce to a P2P Environment

• Use of CloudFIT P2P middleware
  • Support to nodes joining/leaving the network
  • "Bag of tasks" scheduling with speculative execution
  • Full replication of results to ensure high availability
  • Pastry at the overlay network layer
Porting MapReduce to CloudFIT

- Two dependent jobs mimicking Hadoop behavior:
  - MAP – as many tasks as input files
  - REDUCE – as many tasks as computing cores
  - Intermediate data exchange (shuffle/sort)
    - Provided by the replication mechanism of CloudFIT
- Basic API compatibility
Basic experiment

- MapReduce "WordCount"
  - Up to 16 nodes from Grid'5000 testbed
  - Gutenberg Project Science Fiction Bookshelf
- Comparison of Hadoop and CloudFIT-MR
  - Different data sizes
  - Different data blocks
    - 512kB, 1MB, full-file
Performance of MR over CloudFIT

The chart shows the time (in milliseconds) taken to analyse different volumes of data (ranging from 1M to 64M) for various systems:

- CloudFIT-512k
- CloudFIT-1M
- CloudFIT-*M
- Hadoop-512k
- Hadoop-1M
- Hadoop-*M

As the volume of data increases, the time taken to analyse it also increases for all systems, with Hadoop-*M showing the highest time.
The Hadoop framework is composed by two "clusters": MapReduce (computing) and HDFS (file system)

These two clusters were designed to work together

Hadoop Distributed File System
Data is distributed in a hierarchical tree
Replication of data is allowed to improve fault-tolerance

MapReduce
Favor the computing of "local" data
Uncompleted tasks can be executed by any node (speculative execution)
SPoFs on Hadoop

• Hadoop architecture in "trees" present two main Single Point of Failure
  • The JobTracker
  • The NameNode

• Currently, only the NameNode can be replicated

• If the JobTracker hangs, no new jobs can be executed
  • Existing tasks will be completed but no coordination is provided to allow speculative execution
Replicating the JobTracker

- In order to replicate the JobTracker, we develop a snapshot mechanism over Zookeeper
  - Zookeeper
    - Similar to a "shared memory" but with strong replication/coordination properties
  - In our work, we modified Hadoop to replicate the JobTracker state on Zookeeper "database"
Resuming the JobTracker Operation

• The JobTracker makes frequent snapshots
• Hadoop nodes monitor the JobTracker status and keep a list of all connected nodes
  • Nodes removed from the list can be reinserted in the end of the list
• In the case of a failure, the next node in the list installs the last snapshot and resumes the JobTracker operation
  • TaskNodes are redirected to the new JobTracker
A Prototyping Environment

• Experimenting require both a large number of machines and easy management
  • Grid'5000 is sometimes restrictive

• Solution: Docker-Hadoop
  • Container-based virtualization
    • Deployment of dozens of virtual machines in a single node
  • Interface for simplified management of nodes
    • Start/stop of nodes in a click
    • Definition of the roles (JobTracker, etc)
  • Improved reproducibility of experiments
Docker-Hadoop

- https://github.com/vierja/docker-hadoop
Context-Aware Scheduling

- Hadoop was designed for clusters, so only three job scheduler algorithms are proposed by default
  - FIFO
    - The whole cluster is used by each job
  - Fair
    - Designed for batches and small jobs
    - Usually one queue by user
  - Capacity
    - Designed to interconnect several clusters and ensure minimum capacity to each one while allowing the use of exceeding capacity

- Both algorithms use the configuration files as context source, which is not always precise enough
Context-Aware Scheduling

• We improved the Capacity scheduler with context information obtained from system probes
  • CPU speed, CPU load, max and available memory
• Experiments with TeraSort and heterogeneous machines show an improvement on the task allocation
Discussions & Future Works

• P2P implementation
  • Explore different replication strategies to improve performance
  • Better API compatibility with Hadoop
• Fault tolerance on Hadoop
  • Integration on Hadoop 2.x daemon architecture
  • Join of new nodes during a computation
• Context-awareness
  • Improve data locality for the P2P implementation
Conclusions

• MapReduce is one of the leading paradigms for BigData but is also a good paradigm for generic distributed computations (HPC, etc)

• The PER-MARE initiative
  • Porting Hadoop to dynamic environments
    • Context-awareness and Fault-tolerance as key elements
  • Implementing MapReduce on a P2P platform
    • The experiments offer encouraging results
    • Reduced overhead due to a lightweight management of nodes and jobs/tasks