[Considerations regarding MapReduce 2](#_Toc433729028)

[From [Steffenel 2013 - First Steps on the Development of a P2P Middleware for Map-Reduce] 2](#_Toc433729029)

[From [Steffenel et al. 2014 – MR challenges on pervasive grids] 4](#_Toc433729030)

[From [Steffenel et al. 2015 – Leveraging Data Intensive Applications on a Pervasive Computing Plattform : the case of MapReduce] 4](#_Toc433729031)

[From [Steffenel/Pinheiro 2015 – CloudFIT, a PaaS for IoT applications over Pervasive Networks] 5](#_Toc433729032)

[From [Pinheiro 2013 – Requirements for Context-aware MapReduce on Pervasive Grids] 6](#_Toc433729033)

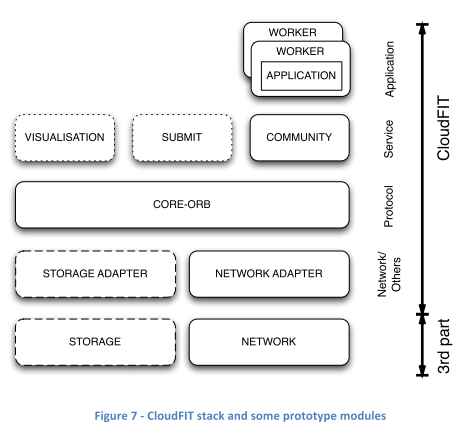
[From [Marozzo et al 2012 – P2P MapReduce: Parallel data processing in dynamic Cloud environments] 6](#_Toc433729034)

[From [Steffenel et al. 2013 – Adaptive Deployment of MapReduce over Pervasive Grids] 6](#_Toc433729035)

# Considerations regarding MapReduce

### From [Steffenel 2013 - First Steps on the Development of a P2P Middleware for Map-Reduce]

* **Basic MapReduce Phases: 2-step procedure**
  1. **Map**: from a set of key/value pairs from the input, the map function generates a set of intermediate pairs: (k1;v1) 🡪 {(k2;v2)}
  2. **Reduce**: from the set of intermediate pairs, the reduce function merges all intermediate values associated with the same intermediate key, so that (k2;{v2}) 🡪 {(k3;v3)}.
* Distr. System: intermediate pairs for a key k2 may be scattered among **several nodes** 🡪 need to gather all pairs for each key k2 so that the reduce function can merge them into the final result
* Additionally:
  + **Splitting** input data among nodes
  + **Scheduling** of jobs’ component tasks
  + **Recovery** of tasks hold by failed nodes.
* **Problems in Hadoop HDFS:**
  + JobTracker JT 🡪 Task Trackers TT(Master Slave, Task Scheduling & Monitoring)
  + Name Node NN 🡪 Data Nodes DN (Master Slave, Data Management)
  + Advantage:
    - Good Performance levels
  + Limitations:
    - **Fault Tolerance** Issues
      * SPOF: JT & NN (crash blocks everything, TT/DN crash not so bad, may be recovered)
    - **Constraints**
      * During execution of a job, **no new node may join**!
        + (due to hierarchical structure of DN 🡪 Overhead in case of restructure of the HDFS tree to be expected)
* Implementation on CONFIIT:
  + **Recovery in case of node failures**:
    - As long as task is not completed, other nodes may pick it up
      * If node fails/leaves grid 🡪 nodes may recover tasks
      * When a node joins, it
        + receives a copy of the working data
        + may pick up available (incomplete) tasks on the shared task list
  + **Support of (re-) connection** (Hadoop only covers node disconnections, but due to tree like structure overhead of restructuring, no (re-)connection while running jobs
  + CONFIIT is a **RING Structure with Token passing!**
  + **P2P** Problem: Intermediate data between map and reduce (for word count) is **bigger** than initial data (annotations about occurrences of a word) **🡪 overload of P2P network**
  + **CONFIIT was abandoned!**
* **Successor of CONFIIT**: **CloudFIT**
  + **Design considerations**: Middleware independent (different P2P), Modularity, Well-defined interfaces, Loose coupling



* Network observer: tries to estimate the number of nodes/cores in the network
* Scheduling in prototype: random (prevents all nodes to execute the same tasks) 🡪 task list is shuffled at each machine 🡪 different task order to execute (no context-aware scheduling of tasks yet)
* End of task execution: task status is updated, task result is broadcasted to other nodes 🡪 **ensures all nodes collaborate**, **avoids duplicate work**
* Results received from other machines will update the task status
* **All tasks finished**: compute job result, store it in resources from storage layer
* **Slowdown against Hadoop** due to overloads of service layer messages (same messages are used to keep nodes updated about task completions and task results 🡪 small data ok, for large data: service layer gets overloaded. Nodes finish by computing most of the tasks locally as few updates are able to reach the other nodes.
  + **Data transfer improvements**:
    - Use **specific data exchange channels** (created on-demand by nodes that wish to complete data sets after receiving update message) 🡪 don’t use service messaging for data transfer…
    - No broadcast of results: split update in two steps:
      * broadcast “COMPLETE” status of task (and ID of node that has results)
      * Acquire task results directly from “server” that has them
      * 🡪 solution is only for FreePastry… maybe different in TomP2P

### From [Steffenel et al. 2014 – MR challenges on pervasive grids]

* **Context Awareness on Hadoop/MR implementations on pervasive grids**
  + **Three categories:** 
    - **Job Schedulers**
    - **task schedule optimizers**
    - **resource placement facilitators** (see references there)
* **Fault-tolerance and volatility/data storage**
* **HDFS** tries to place data for map/reduce phases as close as possible to the processes/tasks that will need it to reduce network access.  **🡪** in **pervasive grid: volatility & speed of resources to prevent data losses** is needed

See also P2P-MapReduce (Marozzo)

Marozzo, F., D. Talia and P. Trunfio, 2010. A Peer-to- Peer Framework for Supporting MapReduce Applications in Dynamic Cloud Environments. In: Cloud Computing: Principles, Systems and Applications, Antonopoulos, N. and L. Gillam (Eds.)., Springer, ISBN 978-1-84996-240-7, pp: 113-125.

Marozzo, F., D. Talia and P. Trunfio, 2012. P2P- MapReduce: Parallel data processing in dynamic cloud environments. J. Comput. Syst. Sci., 78: 1382- 1402. DOI: 10.1016/j.jcss.2011.12.021

* See 5.1 Hadoop schedulers, Context information using standard Java API!!

### From [Steffenel et al. 2015 – Leveraging Data Intensive Applications on a Pervasive Computing Plattform : the case of MapReduce]

* **Problem Hadoop (again)**: **Master-Slave platform for task monitoring/scheduling AND data management on top of hierarchical HDFS**
  + Not adapted to dynamic environments (failure of master may prevent system operation)
* **2 Problems** in “Pervasive grids”: **Data Distribution & Data Processing**
  + **Data distribution**: 2 possibilities
    - **P2P** (peers collaboratively participate to the distribution of the data by exchanging file chunks)
    - **Content Delivery Service** (files are distributed by a secure network of well-known & authenticated volunteers) 🡪 ALSO P2P (e.g. input data from a network of cache peers in P2P ring)
      * Kelley, I., Taylor, I.. A peer-to-peer architecture for data-intensive cycle sharing
  + **Data processing**:
    - **Improve Hadoop performance** with new **Scheduling Algorithms**
      * Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.. Improving mapreduce performance in heterogeneous environments.
      * Chen, Q., Zhang, D., Guo, M., Deng, Q., Guo, S.. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment
      * Ahmad, F., Chakradhar, S.T., Raghunathan, A., Vijaykumar, T.N.. Tarazu: optimizing mapreduce on heterogeneous clusters
    - **Task coordination techniques:** centralized task server, work-stealing/bag of tasks, speculative execution, etc.
      * **Good Performance** only through **minimization of data transfers** over the network!
* **CloudFIT Design**:

### From [Steffenel/Pinheiro 2015 – CloudFIT, a PaaS for IoT applications over Pervasive Networks]

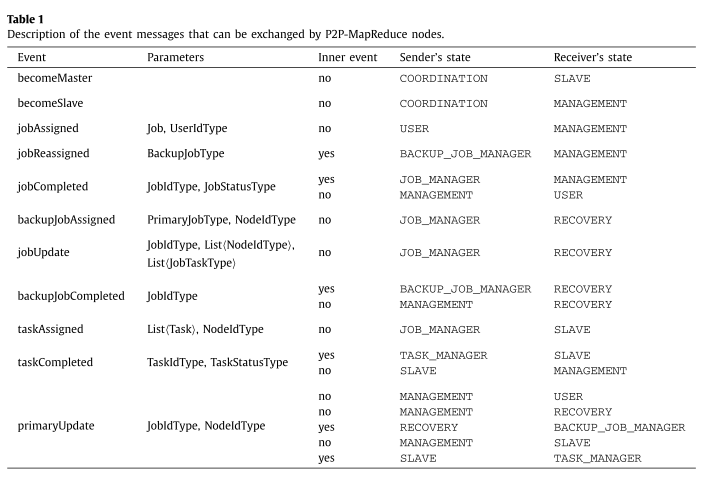
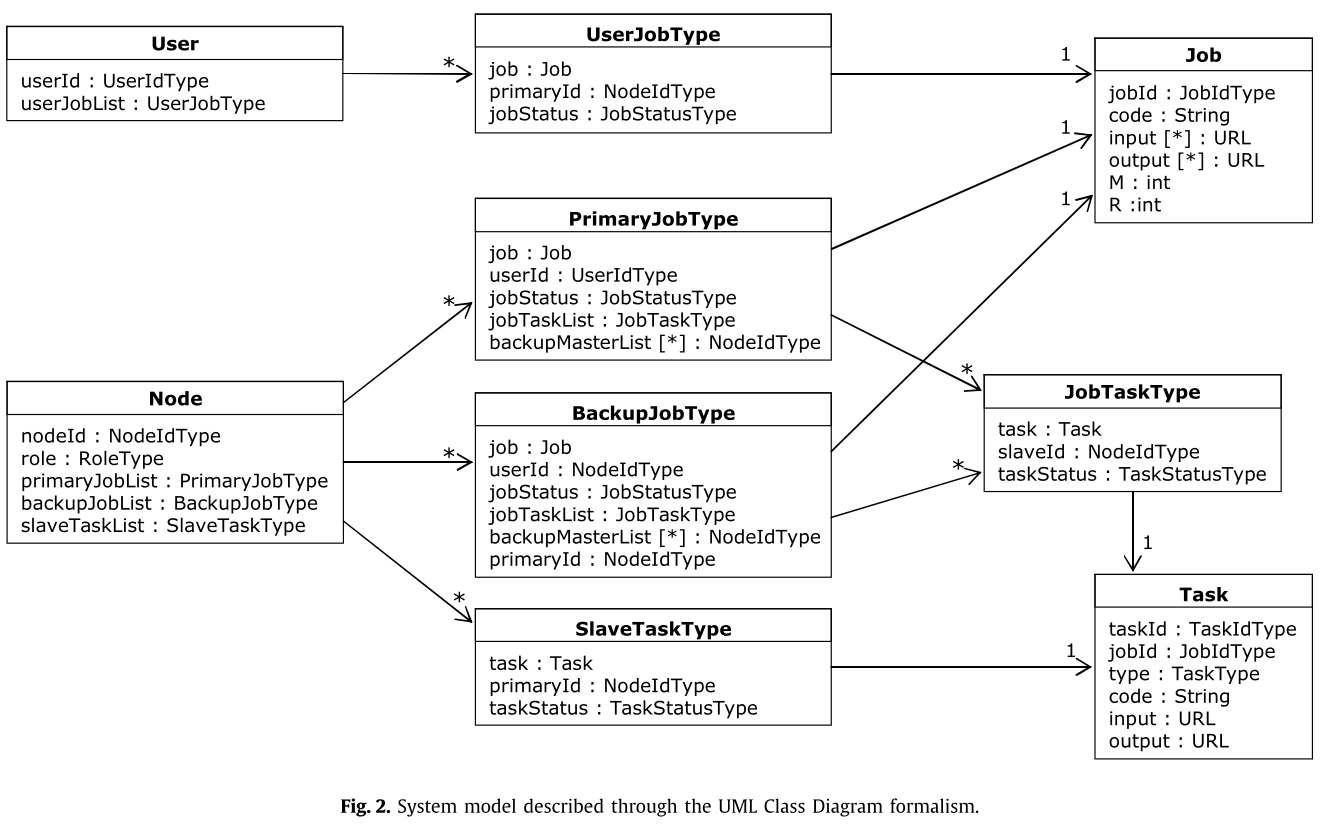
* P2P provides
  + all **communication** & **fault** **tolerance** properties
  + **DHT** storage
* **Scheduling**
  + **Basic Scheduling** 
    - **Randomly rearranges list of tasks at each node** 🡪 helps computation of tasks in parallel without requiring additional communication between nodes (reduces problem of having a task computed by a slow node)
    - Supports **task dependencies** (allows composition of **DAGs**)
    - Improving possible through the use of a **context module** (additional info about **node capacities**)
* **Fault tolerance over**
  + Overlay Network (recovery on leaving/task scheduling on joining)
  + Computing Platform (picking up tasks on node leave/crash/assigning tasks on node join)
* **MapReduce in distributed system**
  + Intermediate pairs (after map) for given key may be scattered among several nodes 🡪 all pairs need to be gathered for each key for merging in reduce phase.
  + Additional features:
    - **Splitting** **data** among nodes
    - **Scheduling** jobs’ component tasks
    - **Recovery** of tasks hold by failed nodes
  + Tasks inside Map and Reduce jobs are independent (while preserving causal relation between Map & Reduce)
  + Several tasks launched during map phase 🡪 produces ki/Vi pairs
  + Each task: a single file/data block is assigned 🡪 independent execution from other tasks
  + On completion: task results are broadcasted to all computing nodes 🡪 each node contains a copy of the **entire** **set** of (ki,Vi pairs) at the end of the map phase
  + **Calculating Map/Reduce tasks**
    - **Hadoop (and CloudFIT)**: guess required number
      * **# Map tasks: == Number of input files**
      * **# Reduce tasks: ==** depends on **dataset size & transitive nature** of data
      * **Result aggregation** possible in CloudFIT and Hadoop **(Combiner)**
  + **CloudFIT Data management/storage/reliability**
    - **Problem of broadcasting large data sets acknowledged (drop n-resiliency)**
    - **Changed it: rely on DHT to perform storage of task results as task\_key/task\_result tuples, Task status messages broadcast keys from each task. 🡪 TomP2P has replication factor k to ensure minimal fault tolerance levels**
  + **Comparisons**
    - Use **immutable objects!!! 🡪 faster**
    - Influences to speed: **(processor type, speed, and) mainly network speed**
      * **Competition between faster & slower nodes**: similar chances to draw a task to execute at the beginning 🡪 faster nodes will complete their tasks first and finally re-execute tasks from slower nodes 🡪 **wastes** **computing** **resources**

### From [Pinheiro 2013 – Requirements for Context-aware MapReduce on Pervasive Grids]

* **Context-awareness:** ability of a system to adapt its operations to the **current context** (aims at increasing usability & effectiveness by taking environmental context into account)
  + **Automatically** **configure** nodes during Hadoop installation (nowadays, its statically using XML files)
  + **Track Node availability (volatility)** arrival and departure of nodes
  + **Task distribution/scheduling** according to nodes availability & current execution context 🡪 node context: availability, network latency, available memory, storage, processor type, etc.
  + **Data placement** (Colocalization in a single node of data & tasks, especially reduce task 🡺 Hadoop’s success reason 🡪 less bandwith usage)
  + **Always consider task distribution TOGETHER WITH data placement**

### From [Marozzo et al 2012 – P2P MapReduce: Parallel data processing in dynamic Cloud environments]

* Related work list lot of other MR implementations (for own related work)
* Zaharia et al: estimate finish times to schedule speculative copies of tasks
* Phoenix: for shared memory systems 🡪 uses threads to spawn parallel map/reduce tasks and shared-memory buffers to facilitate communication (less data copying). Tasks dynamically scheduled accorss available PROCESSORS to achieve load balance & maximize task throughput 🡪 Map reduce also good for multi-core/multi-processor systems
* Their Work:
  + P2P model for clouds
  + At each time, a limited set of nodes is assigned the master role. Other have slave role
  + Role assigned can change over time 🡪 ensures presence of desired master/slave ratio for reliability & load balancing
  + Prevent loss of work in case of master failure:
    - Each master can act as a backup for other masters (primary master for job updates job state on backup nodes (backup masters for job).
      * Avoid overhead: update does not contain whole job information (only changed part)
  + Node can be master or slave… in case of bad master/slave ratio, can be turned into the other to keep good master/slave ratio
  + Nice description of jobs, tasks, users, nodes, see UML
  + Maybe use descriptions as ideas for how to implement different behavior and thinking about RPCs
  + Maybe use description of evaluation for own eval.
  + Maybe see messages for RPCs



### From [Steffenel et al. 2013 – Adaptive Deployment of MapReduce over Pervasive Grids]

* **Nevertheless, good performances can only be achieved through the minimization of data transfers over the network, which is one of the key aspects of Hadoop HDFS filesystem.**
* **Context**
  + **Node availability:** (re-) integrate new/disconnected nodes 🡪 **P2P**
    - Each node has a list of task and associated results
    - Tasks **randomly** **rearranged** at each node 🡪 no additional communication required between nodes
  + **Context distribution mechanism 🡪 P2P**