existing middleware stacks. Figure 1 shows the basic SmartLM scenario for environments where at run-time there is no bi-directional network link available between the license service that created the license token and the remote execution environment, eg due to firewall restrictions.

SmartLM addresses the licensing management issues not only from a technological point of view, but also from the perspective of developing new business models. This approach is necessary in order to convince Independent Software Vendors adopting to the new license technology. More details can be found on the project web pages.

The major part of the licensing technology presented in this article has been designed and implemented prototypically in the European Commissions ICT programme in the FP7 project SmartLM. In the European funded project OPTIMIS (Optimized Infrastructure Services) we will further improve the capabilities of the SmartLM solution, developing additional features, for instance a feature that makes the SmartLM solution more secure in Clouds or extending the capabilities if there is a bi-directional network connection available at run-time.

Links:
elasticLM http://www.elasticlm.com
SmartLM http://www.smartlm.eu/

Please contact:
Wolfgang Ziegler
Fraunhofer SCAI, Germany,
Tel: +49 2241 14 2258
E-mail:
wolfgang.ziegler@scai.fraunhofer.de

Enabling Reliable MapReduce Applications in Dynamic Cloud Infrastructures

by Fabrizio Marozzo, Domenico Talia and Paolo Trunfio

MapReduce is a parallel programming model for large-scale data processing that is widely used in Cloud computing environments. Current MapReduce implementations are based on master-slave architectures that do not cope well with dynamic Cloud infrastructures, in which nodes join and leave the network at high rates. We have designed a MapReduce architecture that uses a peer-to-peer approach to manage node churn and failures in a decentralized way, so as to provide a more reliable MapReduce middleware that can be effectively exploited in dynamic Cloud infrastructures.

MapReduce is a framework for processing large data sets in a highly parallel way by exploiting computing facilities available in a data centre or through a Cloud computing infrastructure. Programmers define a MapReduce application in terms of a map function that processes a key/value pair to generate a list of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Current MapReduce implementations, like Google's MapReduce, are based on a master-slave architecture. A job is submitted by a user node to a master node that selects idle workers and assigns a map or reduce task to each. When all the tasks have been completed, the master node returns the result to the user node. The failure of one worker is managed by re-executing its task on another worker, while master failures are not explicitly managed as designers consider failures unlikely in reliable computing environments, such as a data centre or a dedicated Cloud.

In contrast, node churn and failures – including master failures – are likely in dynamic Cloud environments, such as a Cloud of clouds, which can be formed by a large number of computing nodes that join and leave the network at very high rates. Therefore, providing effective mechanisms to manage such problems is fundamental to enable reliable MapReduce applications in dynamic Cloud infrastructures, where current MapReduce middleware could be unreliable.

At the University of Calabria and ICAR-CNR we have designed an adaptive MapReduce framework, called P2P-MapReduce, which exploits a peer-to-peer model to manage node churn, master failures, and job recovery in a decentralized but effective way, so as to provide a more reliable MapReduce middleware that can be effectively exploited in dynamic Cloud infrastructures.

P2P-MapReduce exploits the peer-to-peer paradigm by defining an architecture in which each node can act either as a master or a slave. The role assigned to a given node depends on the current characteristics of that node, and can change dynamically over time. Thus, at each time, a limited set of nodes is assigned the master role, while the others are assigned the slave role. Each master node acts as a backup node for the other master nodes. A user node can submit a job to one of the master nodes, which will manage it as usual in MapReduce. That master dynamically replicates the entire job state (ie, the assignments of tasks to nodes, the locations of intermediate results, etc.) on its backup nodes. If those backup nodes detect the failure of the master, they will elect one of them as a new master that will manage the job computation using its local replica of the job state.

The behaviour of a generic node is modelled as a state diagram which defines the different states that a node can assume, and all the events that determine transitions from one state to another state (see Figure 1). The slave macro-state describes the behaviour of an active or idle worker. The master macro-state is modelled with three parallel states, which represent the different roles a master can perform concurrently: possibly acting as a primary master for one or more jobs (management); possibly acting as a backup

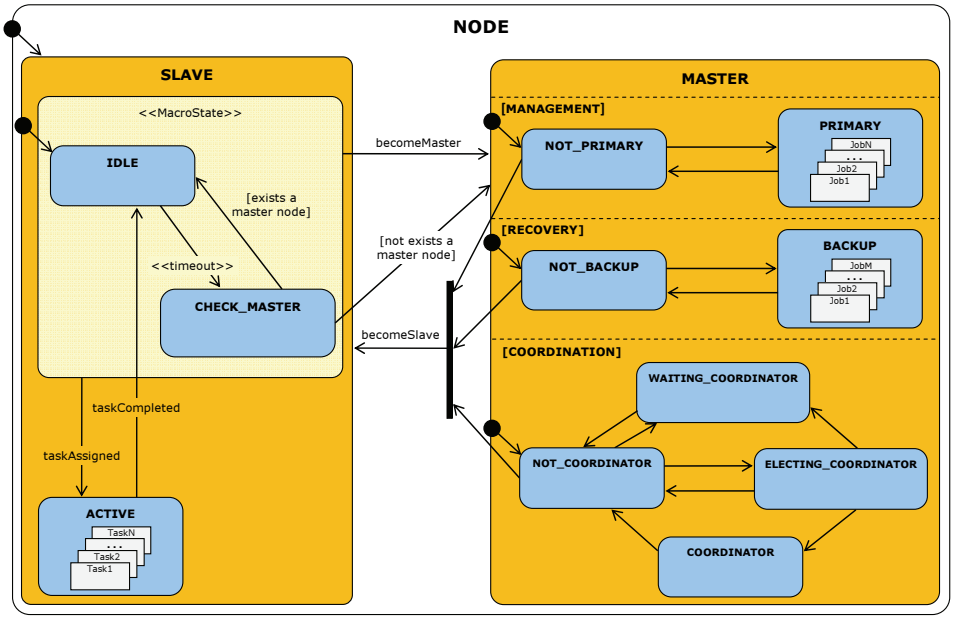

Figure 1: UML State Diagram describing the behavior of a generic node in the P2P-MapReduce framework. The slave macro-state describes the behavior of an active or idle worker. The master macro-state is modelled with three parallel states: Management (the node is possibly acting as a primary master); Recovery (the node is possibly acting as a backup master); Coordination (the node is possibly acting as the network coordinator).

master for one or more jobs (recovery); coordinating the network (coordination). The goal of a master acting as the network coordinator is to ensure the presence of a given percentage of masters on the total number of nodes; to this end, it has the power to change slaves into masters, and vice versa.

We implemented a prototype of the P2P-MapReduce framework using the Sun's JXTA peer-to-peer framework. In our implementation, each node includes three software modules/layers: Network, Node and MapReduce (see Figure 2). The Network module is in charge of the interactions with the other nodes using the pipe communication mechanism provided by the JXTA framework; additionally, it allows the node to interact with the JXTA Discovery Service for publishing its features and for querying the system (eg, when looking for idle slave nodes). The Node module controls the node lifecycle; its core is represented by the FSM component which implements the logic of the finite state machine shown in Figure 1. Finally, the MapReduce module manages the local execution of jobs (when the node is acting as a master) or tasks (when the node is acting as a slave). Currently this module is built upon the local execution engine of Apache Hadoop.

We are carrying out a set of experiments to evaluate the behaviour of the P2P-MapReduce framework compared to a standard master-slave implementation of MapReduce, in the presence of different levels of churn. Early experimental results show that, in contrast to standard implementations, the P2P-MapReduce framework does not suffer from job failures even in presence of very high churn rates, thus enabling the execution of reliable MapReduce applications in very dynamic Cloud infrastructures.

Links:

http://labs.google.com/papers/mapreduce.html

https://jxta.dev.java.net

http://hadoop.apache.org

Please contact:

Domenico Talia
ICAR-CNR and
DEIS, University of Calabria, Italy
Tel: +39 0984 494726
E-mail: talia@deis.unical.it

Fabrizio Marozzo and Paolo Trunfio
DEIS, University of Calabria, Italy
E-mail: fmarozzo@deis.unical.it, trunfio@deis.unical.it

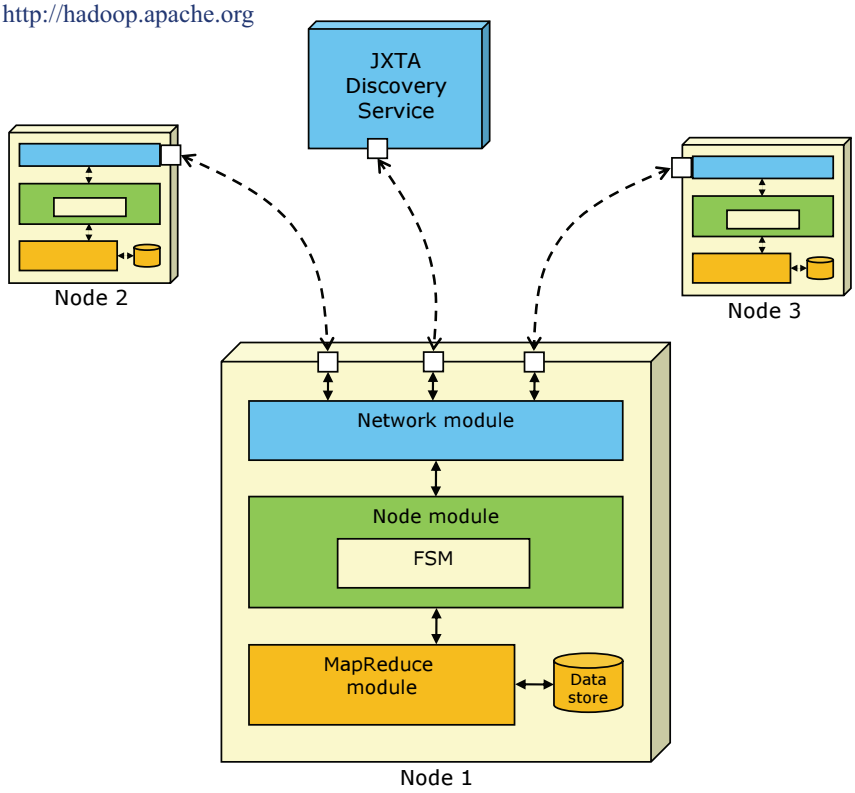

Figure 2: Software architecture of the P2P-MapReduce framework. Each node includes three software modules/layers: Network, Node and MapReduce. The Network module provides communication mechanisms with the other nodes and with the JXTA Discovery Service. The Node module implements the logic of the finite state machine shown in Figure 1. The MapReduce module manages the local execution of jobs and tasks.