



IP-FP6-015964

AEOLUS

Algorithmic Principles for Building Efficient Overlay Computers

Deliverable D6.2.10

Extended and improved prototype of a Web Computing Library

Responsible Partner: University of Paderborn (D)
Report Preparation Date: September 2008

Contract Start Date: 01/09/05 Duration: 48 months
Project Co-ordinator: University of Patras (EL)

D6.2.10 is a software deliverable concerning the extended and improved version of our peer-to-peer based web computing library (PUB-Web) which allows to execute tightly coupled, massively parallel algorithms in the bulk-synchronous (BSP) style on PCs distributed over the internet, utilizing only the donated idle times on these PCs. Anyone with a computer and an internet connection can participate in this system. To do so, one has to install also the software of the AEOLUS overlay computing platform.

During the third year of AEOLUS we have extended PUB-Web by some functionalities based on the software developments of several partners. These extensions are: DHHT load balancer developed by UPB, JCache benchmarking library developed by UNIPD, Mimic job-scheduler developed by CAU, STA timestamping mechanism developed by UNISA and SybilGuard reputation service developed by KULeuven.

The motivations to integrate these functionalities into PUB-Web are the following: JCache gives us the opportunity to exploit the characteristics of the computing nodes for load-balancing (DHHT) and scheduling (Mimic) decisions. STA is a security mechanism that supports the fault-tolerance feature of PUB-Web by timestamping BSP-processes as well as verifying the timestamps of backup copies of crashed BSP-processes. SybilGuard is a reputation service which supports the authentication mechanism of PUB-Web. In the following we will briefly describe only the idea behind the DHHT load balancer. For a detailed description of the other functionalities we refer to the corresponding deliverable D6.2.8.

When utilizing the unused computation power in a web computing environment, one has to deal with unpredictable fluctuations of the available computation power on the particular computers. Especially in a set of tightly coupled parallel processes, one single process receiving little computation power can slow down the whole application. A way to balance the load is to migrate these “slow” processes. The extended and improved version of our web computing library now contains an implementation of our DHHT-based load balancer (cf. Deliverable D3.2.5).

At the beginning all computing nodes in the system are hashed uniformly at random into the $[0; 1]$ -interval, which is interpreted as a ring but in the demo depicted as a line with wrap-arounds for simplicity of illustration. There is a line associated with each node, whose gradient is the inverse of the currently available (globally normalized) computation power (and the line is perpendicular to the x-axis if the node has no computation power available). The lower envelope for the $[0; 1]$ -interval is defined, at any point p , as the node whose line is the lowest at point p . We use this lower envelope to assign sub-ranges of the $[0; 1]$ -interval to the nodes. Finally, all BSP processes to be executed are also hashed uniformly at random into the $[0; 1]$ -interval and must be run by the nodes associated with them through the lower envelope.

Although the continually fluctuating available (donated) computation power is the most important resource to fairly share among all parallel processes, there are also other criteria to consider such as the network bandwidth, for example, as BSP programs communicate a lot. If a BSP program would be scheduled entirely on a fast connected component of the network although there would be a few faster processors available outside this component,

it will run faster due to the reduced networking delays than it would have run when just scheduled according to processing power.

Thus, we aim at clustering the PUB-Web network according to bandwidth. Typically, computers either form a fast connected subnet, which in turn is connected to the Internet via a less powerful link, or they are isolated machines directly connected to the Internet. In either case, the link to the Internet is the component with the least bandwidth of a network path. The backbone of the Internet has such a high bandwidth that we can model it as a single central node, to which all local subnets of computers and isolated machines are connected with an edge whose weight correlates to the bandwidth of this Internet link.

The goal now is to cluster the PUB-Web network such that all local subnets whose size is above a suitable threshold form its own cluster, and all other small subnets and isolated computers build a number of clusters such that, within each cluster, the bandwidth is as homogeneous as possible and that the cluster size fits between suitable upper and lower bounds. We are aware that there exists a broad variety of clustering algorithms, from which we could pick the most suitable one to run on a specialized node; but since PUB-Web is a highly dynamic peer-to-peer network, we have started to investigate in a fault-tolerant, adaptive, and scaling distributed clustering algorithm.

Since the number of computing nodes is so much larger than the number of processes expected per BSP program in average, the overhead of maintaining the DHHT data structures is unfavorable with a growing size of the PUB-Web network compared to the benefit gained by obtaining slightly better schedules on a bigger set of nodes to choose from. Thus, we cascade two levels of DHHT load balancing: On the top level, a DHHT load balancer assigns BSP programs (i.e. groups of processes) to fast connected clusters of the PUB-Web network. On the lower level, one DHHT load balancer per cluster schedules the BSP processes on the particular computing nodes within the cluster. Finally, we need a ready-queue to buffer BSP programs induced into the PUB-Web network in order to avoid overloading the system (note that our load balancer would still generate fair schedules in the overload case, but the overall performance would slow down due to insufficient main memory on the computing nodes).

The source code together with a slide presentation and instructions for running it are available at <http://aeolus.ceid.upatras.gr/files/AEOLUS-DEMO-CD.rar>