Combining Metacomputing and High Performance Computing

Nabil Abdennadher
GRIP Research Group - DI - EPFL
Swiss Federal Institute of Technology
CH-1015 Lausanne, Switzerland

Gilbert Babin
Département d'informatique
Université Laval
Québec, Canada G1K 7P4

Pierre Kuonen
Groupe de compétences Infotronique
Haute école valaisanne
CH-1950 Sion, Switzerland

Abstract ***Current tools available for High Performance Parallel/Distributed computing require that all the computing nodes be known in advance: each computer involved in the execution must be properly configured and the execution environment must usually know where the different modules of the parallel program will be executed. In this paper, we describe how the Web Operating System (WOS^{TM}) environment may be used to dynamically locate available resources on a parallel/distributed environment, particularly on the the Swiss-Tx parallel computer. We also details how the different modules of the parallel program are assigned to the different processors of the Swiss-Tx parallel machine.***

Keywords: High Performance Computing, Metacomputing, Web Operating System (WOS^{TM}), Mapping Algorithm, Resource localization.

1 Introduction

Until recently, High Performance Computing (HPC) mainly involved solving huge numeric problems using matrix calculations on SuperComputers specially designed for this purpose. Access to these SuperComputers was often cumbersome, preventing non expert end-users to easily exploit them. The future of centralised supercomputers depends to a large extent on the development of interfaces for accessing them from the user's desktop in a uniform and user-friendly manner.

Moreover, advances in networking technology and computational infrastructure are changing the HPC landscape. Tightly coupled, dedicated processors are replaced by loosely coupled independant machines connected via standard local or wide area networks. HP Computers are no longer seen as isolated SuperComputers used to solve specific numeric problems, but rather as a "node" belonging to a confederation, tied together by a network and contributing in the resolution of the user's problem. This underlying confederation is commonly called a *metacomputer*. Furthermore, centralized High Performance (HP) applications developed with proprietary, closed-source, hardware-dependant environments are more and more often replaced by distributed "components" sharing and managing resources spread over the networked environment.

In this context, the present research is motivated by the need to coalesce the power of distributed systems into a single metacomputer. This metacomputer can be used to run applications which require resources exceeding those available on any single available platform. The

search, selection, configuration and exploitation of these resources must be carried out in a uniform and user-friendly manner. This article presents work done to integrate the Swiss-Tx HP computer prototype, developed at EPFL, in the Web Operating System, WOS™ metacomputing environment developed in part at Université Laval. The goal is not only to provide an "ergonomic" interface to access the Swiss-Tx resources, but also to consider this machine as a "simple" WOS™ node among others and to integrate it into the WOS™ community.

This paper is organized as follows. Section 2 provides a definition of metacomputing and services that it can provide, and details the structure and functionalities of the WOS™. Section 3 describes what metacomputing, and in particular the WOS™ environment, can bring to HP computing. Section 4 describes the architecture of the Swiss-Tx machine. Section 5 describes how to integrate the Swiss-Tx machine into the WOS™ community. Finally, section 6 provides additional information regarding the status of the project and concludes the paper.

2 Metacomputing

We share Buyya's [1] definition of a metacomputer : a set of computers sharing resources and acting together to solve a common problem. Our vision of a metacomputer comprises many computers and terabytes of memory in a loose confederation, tied together by a network. The user has the illusion of a single powerful computer; he manipulates objects representing data resources, applications or physical devices.

At this point, it is important to distinguish between a parallel computer and a metacomputer. The main difference is the behavior of the computational nodes. Contrary to a parallel computer, a metacomputer is a dynamic environment that has some informal pool of independant nodes, each relying on its own complete operating system, and which can join or leave the environment whenever it desires. A metacomputer is distinguished from a simple collection of computers by a software layer (middleware) which transforms a collection of independant resources into a single, virtual and coherent machine.

To better understand what metacomputing is, we first introduce the concept of Grid Computing, which may be compared to the electricity grid [2] : when we power up our computer or television, we don't care about the location of the electricity generator that effectively provides energy to the concerned appliance. In the same manner the national electricity grid routes electricity across hundred of miles, the grid computer, i.e. a set of connected supercomputers, should allow the transparent execution of a program by searching and allocating the resources it requires. It is the grid computer's responsability to support transparent security, scheduling, data displacement, fault tolerance, conversion, etc. Metacomputing is a generalization of Grid Computing where the supercomputers may be replaced by off-the-shelf computers.

2.1 Metacomputing Services

A metacomputer should provide four basic services [3] :

1. *Transparent Remote Execution.* By using a metacomputer, a user should be able to execute his application by simply typing a command line. The system should select the appropriate node(s) among those the user is allowed to use, transfer binary code and, launch execution. The transfer of the input data and the storage of the output data should be done transparently by the metacomputer. Finally, the user should not know about the queuing system of the selected executing node(s).

2. *Transparent Access Distributed File System.* A metacomputer should allow transparent access to a file, regardless of its location. NFS is a well known exemple

of a distributed file system [4]. However, NFS requires super-user configuration. The World Wide Web is another well known distributed file system limited to read-only access.

3. *Wide-Area Parallel Processing.* The goal is to execute a single parallel application by using multiple remote resources (parallel machines). The application should tolerate the latency involved by the transfert of data between the remote sites. Problems, well known in parallel processing, such as task assignment, load balancing and fault tolerance should be taken into account by metacomputing.

4. *Meta-Applications.* Meta-Applications are composed of a set of connected legacy applications that were previousely executed as standalone applications. The output of the first application is the input of the second, etc. The meta-application can thus be represented by a graph, where nodes are stand-alone applications and edges are data communication links between nodes. Beside the scheduling problem already encountered above, there is another issue : data are geographically distributed and it is the metacomputer's responsability to decide when to migrate computation to the data or vice versa.

2.2 The Web Operating System

The Web Operating System (WOS™) [5] is a metacomputing environment developed to provide a user with the possibility to submit a service request without any prior knowledge about the service (where it is available, at what cost, under which constraints) and to have the service request fulfilled within the user's desired parameters (time, cost, quality of service, etc.). Three features make the WOS™ an attractive environment for metacomputing:

1. *Open Access.* Most of the metacomputing projects, such as Globus [6], Legion [7, 3], and NetSolve [8], require login privileges and a global catalog of resources. This may be interesting for small networks but could be impractical for large ones. Contrary to this, the WOS™ uses distributed databases, called warehouses, that allow open access and search procedures. Each WOS™ node has its own local warehouse which contains all the resources that it can provide. These resources can be hardware or software. The search engine takes into account the dynamic nature of the Web. The WOS™ is based on a demand-driven computation model: users' queries are only processed when needed and prior results are stored in the warehouses, where they can be accessed later on.

2. *Universality.* The WOS™ aims to supply users with adequate tools that allow the implementation of specific services not initially foreseen. In order to achieve this goal, a generic service protocol (WOSP), provided by the WOS™, allows the WOS™ node administrators to implement a set of services, called a *service class*, dedicated to specific users' needs. WOSP is in fact a generic protocol defined through a generic grammar [9]. A specific instance of this generic grammar provides the communication support for a service class of WOS™. This specific instance is also referred to as a *version of WOSP*; its semantics depends directly on the service class it supports. In other words, knowing a specific version of WOSP is equivalent to understanding the semantics of the service class supported by that version. Several versions of WOSP can coexist on the same WOS™ node.

3. *Intensionality.* The WOS™ manages contexts: hardware, software, time, place, etc. The basic nature of the WOS™ is to answer users' requests while considering all these contexts; the WOS™ node will provide the best resources available, as a function of the current context, which always changes.

3 Metacomputing for High Performance Computing

The approach we propose is to take advantage of some services provided by metacomputing environments and to apply them during the configuration phase of HP parallel applications. The most important service is the mapping of the parallel application into the target machine. In order to garantee this service, the metacomputing environment should be able to localize the resources required by the different modules of the parallel application. In the context of WOS^{TM} , the configuration of HP parallel applications is a specific service that should be supported by a specific version of WOSP, namely *HP-WOSP* [10].

A parallel/distributed application is usually represented by a graph where vertices are sequential processes and edges communications between the processes. These processes are assigned, during execution, to a single processor. Let's assume that, instead of representing sequential processes, vertices represent processes that can be split into a set of parallel/distributed *modules* which can themselves be modeled by a graph. Each vertex of this new graph can be recursively decomposed, until all the vertices represent atomic sequential processes. This recursive decomposition of a parallel/distributed application can be represented by a tree where the root represents the entire application and the leaves the elementary sequential communicating processes. The intermediate vertices are some aggregation of the elementary sequential processes. Edges between a given vertex and its children represent the decomposition process. This tree is called the *granularity tree* (GT). For some parallel languages, the GT can be automatically produced, whereas in other cases it can be manually produced by the programmer using an adequate description format.

Each vertex of the GT must indicate the resources it needs for its execution. These resources belong to one of the three following families :

1. *Software resources.* Modules that constitute the parallel application are not necessarily developed within the same environments. Each module may need a particular software resource such as a specific execution environment : Java, PVM, MPI, Corba, etc.

2. *Hardware resources.* Some modules may require a specific hardware CPU architecture, a particular network topology, a minimum main memory, or other particular hardware resources.

3. *Virtual resources.* Some modules may require a minimum threshold of CPU performance or a minimum bandwidth or latency time of the network to be used for communication, etc. The expression of these "virtual" resources is very important in the context of High Performance Computing.

We assume that the target machine for the parallel/distributed application is a network of WOS^{TM}nodes ranging from regular desktop workstations to HP parallel computers. Using the GT, the mapping phase consists in assigning modules to the target machine by allocating a subset of the GT vertices to the target machine. This subset must represent the whole application. During the assignment of each vertex, it will be checked whether the selected WOS^{TM}node is able to provide all the necessary resources needed by the given vertex. In particular, the mapping must take into consideration the current workload of the WOS^{TM}nodes and the traffic over the network, in order to meet the performances (virtual resources) required by the corresponding vertex being assigned.

4 The Swiss-Tx Computer

The Swiss-Tx (The Swiss Teraflops Supercomputer) project was proposed to show that it is feasible to construct a low cost, massively parallel supercomputer achieving a computational speed of at least one Teraflop. The ma-

chines developed within this project are constructed with commodity parts available from the market, in hardware (PC or workstation boards including processors, memory, disks, interconnects) and software (operating system, compilers, libraries). Each machine consists of a certain number of nodes connected with a fixed topology communication network and each node consists of few boards (typically from 4 to 8) connected with a full crossbar. The communication protocol is based on an original zero-copy, low latency, high bandwidth concept named T-NET and developed at the small-size high-tech Swiss company SCS AG located in Zürich. The Swiss-Tx supercomputer is the "long arm" of a PC or of a workstation with the same board architecture and the same operating system.

In order to build a teraflop metacomputer, several intermediate prototypes have been realized. The most recent version of these prototypes, the Swiss-T1 [11]), has been installed at EPFL in January 2000. The computing part of the Swiss-T1 consists of 32 *boxes* built using one bi-processor Compaq 667 MHz DS20e workstations. These 32 boxes are distributed on eight *nodes* , each containing four boxes. These eight nodes are connected together using eight 12x12 crossbars. On each crossbar eight ports are used to connect the boxes, the four remaining ports are used to connect the crossbars to other nodes using a 2-Ring topology [12]. In addition to the T-NET network, all boxes are connected together and to the "rest of the world" through two fast Ethernet and one gigabit Ethernet switch. The computing part of the Swiss-T1 contains in total 64 Alpha DS20e processors and each boxes contains 1 Gbytes of main memory and 13 Gbytes of local disk space. The computing part is completed by a front-end part connected through the Ethernet switches and consisting of 2 bi-processor Compaq DS20e workstations connected to two RAID storage units. The two workstations of the front-end part are connected together with a crossbar switch. Figure 1 shows the complete configuration of the Swiss-T1 prototype, each box runs a standard DEC-Unix operating

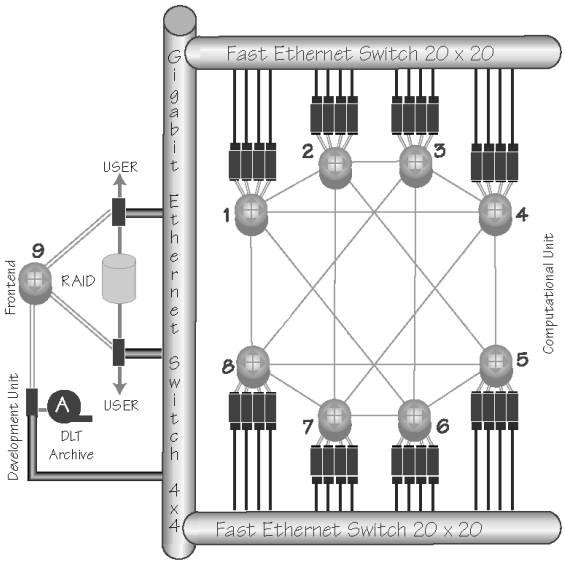

Figure 1: Hardware configuration of the Swiss-Tx prototype

system.

5 Configuring HPC Applications on the Swiss-Tx

We assume that the parallel application to execute is represented by its granularity tree, constructed manually by the developer. For some specific homogeneous environments, the granularity tree can be generated automatically and dynamically by a compiler.

As a starting point, the user specifies parameters and constraints to execute the parallel program, for example, date and time of execution, etc. From that point on, the control is passed on to the WOS™.

The WOS™ performs the following tasks, if needed :

1. *Locate WOS™nodes.* This occurs only when an insufficiant number of WOS™nodes that can perform parallel computing are locally known. In this case, the WOS™will use a discovery/location protocol (WOSRP) to identify new nodes that understand the version of WOSP used for High Performance Computing (HP-WOSP).

2. *Locate nodes that can participate in the current request.* In this step, we want to identify nodes that can be used to answer the user's specific needs and requirements. We use the search approach developed for the WOS™ [13]. Since one execution may require different resource sets, the search parameters will provide one set of parameters for each of these distinct resource sets. When it receives such a query, a WOS™ node will identify all the resource sets it can provide. The reply constructed by each requested node only includes the resource sets it can provides.

3. *Collect replies from all the nodes.* The requesting WOS™ node collects all replies. At this point, it will determine which nodes would be asked to participate in the execution. Note that it may be necessary to launch additional search steps (1 and 2 above) to complete the configuration.

4. *Assign the granularity tree nodes to WOS™ nodes.* The node selection is made from the top-down. The root node of the granularity tree represents the whole parallel/distributed application. It also contains information about the resources required for its execution (*e.g* CPU performance, network characteristics, execution environment, etc). The mapping algorithm identifies a WOS™node which can provide these resources. If the search process succeeds (*i.e* a node which satisfies these "constraints" is found), the whole application is assigned to the selected WOS™node. If the search process fails, the mapping algorithm proceeds to the assignment of the children vertices. Since these children are the result of the decomposition operation, the performances they request are less "important" than those needed by their ancestor. Thus, the probability to find a WOS™node providing these performances is higher. This recursive process is repeated until each covered vertex is allocated to a WOS™node.

   In the context of the Swiss-Tx machine, the mapping algorithm should try to assign a vertex to the same processor. If this assignment is not possible, children vertices are assigned to the same Swiss-Tx box (bi-processor workstation). Communications between siblings are then achieved within the same shared memory. If this assignment is not possible, children vertices are assigned to processors belonging to the same Swiss-Tx node. Communications are then achieved by the local crossbar. Finally, if the siblings are assigned to processors belonging to two or more different Swiss-Tx nodes, communications will pass through at least two crossbars.

5. *Reserve the resources.* Here, as a result of step 4, the WOS™ simply indicates to the selected nodes that it will use a certain set of resources, based on the information received. A node can still reject or accept a reservation request.

6. *Prepare the resources* In this last step, all remotely located resources required for the execution are transmitted (e.g., data files, program source files, program binary files). Then all resources requiring local preparation are processed (e.g., compiling source files, linking binary files, launching particular execution environment such as PVM, Java, etc.). Finally, the application can be launched without any interference with the WOS™.

The search results are preserved in the local warehouses. This way, subsequent executions with the same (or similar) parameters will reuse the results rather than perform the whole search again.

6 Conclusion

This article has discussed how the WOS™ environment could be adapted to configure High Performance applications on a metacomputer that integrates the Swiss-Tx machine.

More specifically, a detailed description of how WOS™ nodes should interact was presented. We also discussed how the proposed version of WOSP can dynamically select the most appropriate execution environment for the parallel program to be run. This new service class enables the support of Wide-Area Parallel Processing within the WOS™ environment. The search method adopted can satisfy the constraints imposed by the user. Finally, it is important to point out that, altough the WOS™ allows for the dynamic selection of the execution environment most suitable for a specific application, within specific constraints, it is not involved in the execution stage. As a consequence, an application could take advantage of a new, possibly more efficient execution environment without any modification of the WOS™itself. We plan to implement the HP-WOSP on the Swiss-Tx machine, integrate it into the WOS™ environment, and conduct a significantly large series of experimentations in the near future.

References

[1] R. Buyya. *High Performance Cluster Computing : Architectures and Systems*, volume 1. Prentice Hall PTR, Upper Saddle River, N.J., USA, 1999.

[2] I. Foster and C. Kesselman, editors. *The Grid : Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, USA, 1999.

[3] G. Lindahl, A. Grimshaw, A. Ferrari, and K. Holcomb. Metacomputing — what's in it for me ? http://legion.virginia.edu/papers/why.pdf, last visited on Jan. 20, 2000, 1998.

[4] E. Levy and A. Silberschatz. Distributed file systems: Concepts and examples. *ACM Comp. Surveys*, 22(4):321–374, Dec 1990.

[5] P. Kropf. Overview of the WOS project. In *1999 Advanced Simulation Technologies Conferences (ASTC 1999)*, San Diego, CA, USA, Apr 1999.

[6] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int'l J. of Supercomputer Applications*, 1997.

[7] A.S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver, and P.F. Reynolds. The Legion vision of a Worldwide Virtual Computer. *CACM*, 40(1), Jan 1997.

[8] H. Casanova and J. Dongarra. NetSolve: A network server for solving computational science problems. *Int'l J. of Supercomputer Applications and High Performance Computing*, 3(11):212–223, 1997.

[9] G. Babin, P. Kropf, and H. Unger. A two-level communication protocol for a Web Operating System (WOS™). In *IEEE Euromicro Workshop on Network Computing*, pages 939–944, Västerås, Sweden, Aug 1998.

[10] N. Abdennadher, G. Babin, P. Kropf, and P. Kuonen. A dynamically configurable environment for high performance computing. In *High Performance Computing 2000 (HPC 2000)*, Washinton, DC, USA, Apr 2000.

[11] P. Kuonen and R. Gruber. Parallel computer architectures for commodity computing and the Swiss-T1 machine. *EPFL-Supercomputing Review*, (11):3–11, Nov 1999.

[12] P. Kuonen. The k-ring: a versatile model for the design of mimd computer topology. In *High-Performance Computing Conference (HPC'99)*, pages 381–385, San Diego, USA, 1999.

[13] H. Unger, P. Kropf, G. Babin, and T. Böhme. Simulation of search and distribution methods for jobs in a Web operating system (WOS™). In A. Tentner, editor, *High Performance Computing Symposium '98*, Boston, MA, USA, Apr 1998.