Published in IEEE/ACM International Symposium on, 568-573, 2001
which should be used for any reference to this work

A WOS™-Based Solution for High Performance Computing

Nabil Abdennadher
University of Applied Sciences
Geneva, Switzerland
abdennad@eig.unige.ch

Gilbert Babin
Information Technologies
HEC — Montréal
Montréal, Canada H3T 2A7
Gilbert.Babin@hec.ca

Peter Kropf
Dép. informatique et rech. op.
Université de Montréal
Montréal, Canada H3C 3J7
kropf@iro.umontreal.ca

Abstract

Most of the development environments for High Performance Parallel applications require that all the computing modules and resources be known in advance. The execution environment must know where the different program modules will be executed, and must properly configure each computer involved in the execution. In this paper, we describe how the Web Operating System (WOS™) environment may be used to dynamically adjust the granularity of parallel programs, locate available computers to perform the computations and how these computers are dynamically configured. The WOS [7] is a metacomputing environment suitable for supporting and managing distributed/parallel processing on wide and local networks. Communication between WOS nodes is realized through a generic service protocol (WOSP) and a discovery/location protocol (WOSRP). WOSP may be versioned to support specialized services. In this paper, we focus on the design of two such versions for Parallel/Distributed applications and High Performance computing. These versions support the location and setup of computational nodes for these applications.

1. Introduction

Advances in networking technology and computational infrastructure changed the High Performance Computing (HPC) landscape. Tightly coupled, dedicated processors tend to be replaced by loosely coupled independent machines connected via standard local or wide area networks. Centralized *H*igh *P*erformance (*HP*) applications developed with proprietary, closed-source, hardware-dependant environments are more and more replaced by distributed "components" sharing and managing resources spread over a networked environment. The new HP distributed platforms are accessed from the user's desktop in a uniform and user-friendly manner, such as provided by the Web's interfaces. The network environment combines multiple administration domains, heterogeneous computing platforms and security policies. Sharing and managing the resources spread over this network therefore becomes a cumbersome task. This problem is called the wide-area computing problem [10].

The wide-area computing problem can be solved in an *ad hoc* manner for each application: scripts and various network tools can serve for this purpose. However, these solutions are very limited, lack scalability, and require a specific knowledge of the architecture of the machines.

A more systematic way to solve this problem is to build a Network Operating System (NOS) for the management of distributed execution environments. This NOS would provide high level means for sharing and managing complex resources distributed over the network. We think that metacomputing is one promising approach to reach that goal. The purpose of metacomputing is to give the illusion of a single machine by transparently managing data movement, scheduling of application components on available resources, fault detection, and protection of user's data and physical resources.

However, requirements for HPC go far beyond transparent management and use of resources distributed over the network. In the context of HPC, the metacomputing environment must meet the performance requirements of the application from a computational and communication standpoint. To achieve this goal, several metacomputing environments support HPC by providing their own, closed-source, HP execution tools. We argue that, although this approach favours transparency, it does so at the expense of portability and efficiency. It binds the user to the specific HP execution tools supported by the metacomputing environment selected.

The approach proposed in this article follows two primary objectives:

1. Satisfy the HP constraints of a given application by using parallel and distributed computation and,

2. Specify very little about implementation. This goal should be realized by using metacomputing tools during the configuration of the HP application (searching, reserving and assigning resources to the application).

To meet these objectives, we select the Web Operating System (WOS™) [7] as a metacomputing environment. The WOS can be seen as a collection of service classes. Our specific contribution consists in two new service classes.

The first service class, called RD-WOSP, is used to assist users during the configuration step by searching and locating the most suitable resources for the execution of their Parallel/Distributed applications (*PD* applications). Resources can be hardware (CPU, memory, network, etc.) or software (compiler, interpreter, library, etc.). We assume that a PD application is composed of a set of heterogeneous software modules. Each module requires a set of resources which can be software or hardware.

The second service class, called HP-WOSP, is an extension of RD-WOSP service class. It aims at taking into account the High Performance constraints of a particular HP application. These constraints are expressed by the user in terms of termination deadline, effective CPU performance, etc. In this context, a HP application is a sequential or PD application subjected to HP constraints.

This article is organized as follows : Section 2 provides a definition of metacomputing. Sections 3 describes the architecture of the WOS. Section 4 and 5 respectively describe the two service classes: RD-WOSP and HP-WOSP. Section 6 provides additional information regarding the status of the project and concludes the paper.

2. What is Metacomputing ?

A metacomputer is a set of computers sharing resources and acting together to solve a common problem given by the user [3]. A metacomputer may comprise many computers of any kind 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 being the behavior of the computational nodes. A metacomputer is a dynamic environment that has some informal pool of independent nodes, each relying on its own complete operating system, and which can join or leave the environment whenever it desires. According to this definition, some parallel computers, such as the IBM SP series or the Swiss-T1 machine [9] can be considered as local metacomputers, which is not the case for the Cray T3D. In addition, a metacomputer is distinguished from a simple collection of computers by a software layer (middleware) whereas this transforms a collection of independent resources into a single, virtual and coherent machine.

A number of research projects have produced useful metacomputing tools. The best known projects are Globus, Legion, and WebOS. Globus (http://www.globus.org) is a large US based project which provided the fundamental technology needed to build grid environments [6]. Globus comprises a set of modules that implements high-level services used in a computational grid environment. Each module defines an interface which is used to invoke that module's mechanisms.

Legion (htp://www.legion.virginia.edu) is an object-oriented metacomputing environment, intended to connect thousands of hosts ranging from PC to massively parallel supercomputers. One of the primary objectives of Legion is high performance via parallel computation. Indeed, Legion is based upon a parallel computing model. It is basically a set of interfaces for an object system. The prototype developed at University of Virginia is one implementation that exhibits those interfaces.

WebOS [12] is one attempt to easily access geographically distributed resources. Specifically, it deals with the use of CPU, RAM, and disk space of resources scattered across the Internet. The approach adopted in that project is a transposition of classical operating systems problems to the reality of the Internet. The solutions proposed are highly coupled to the operating system. We can also point out that a global catalog of available resources is necessary.

Other examples of metacomputing environments are: NetSolve [5] developed at University of Tennessee, Milan which is a joint effort between New York University and Arizona State University, Unicore, Paraweb, Charlotte, etc. Details about these projects and others can be found in the Web site http://www.computingportals.org.

3. The Web Operating System

The Web Operating System (WOS) [7, 2] was developed to provide a user with the possibility to request a service without any prior knowledge about the service (where it is available, at what cost, under which constraints) and to have that service request fulfilled within the user's desired parameters (time, cost, quality of service, etc.). Each WOS node has to fulfil the task of server as well as of client. A WOS client requests the execution of a service selected by the user sitting in front of his machine, while the WOS server accepts or rejects requests for a service. Two features make the WOS an attractive environment for metacomputing: *Open Access* and *Universality*.

3.1. Open Access

Most of the metacomputing projects, such as Globus and Legion, require login privileges and a global catalog of resources. This may be interesting for small networks but could become impractical for large ones. In contrast to this, the WOS uses distributed databases, called warehouses. Every WOS node maintains a local warehouse which holds information about the set of resources located on it. A local

warehouse can also contain informations about location and state of remote resources [11]. For instance, the warehouse stores information about other nodes that may be part of a metacomputer. Furthermore, that information is collected automatically and dynamically. In fact, a WOS warehouse is more than just a static database with a limited storage amount; each warehouse in the WOS must have the ability to decide without any additional user activity, which information should be stored in which place in the warehouse, replaced or deleted, obtained from an other warehouse and which one, checked or replaced by new information or *a priori* collected information.

3.2. Universality

The WOS aims to supply users with adequate tools that allow for the implementation of specific services, anticipated or not. WOS provides users and class implementors with great flexibility in the semantics of their application. In order to achieve this goal, a generic service protocol (WOSP), provided by the WOS, allows the WOS node administrators to implement any set of services, i.e. *service classes*, dedicated to any specific user needs. WOSP is in fact a generic protocol defined through a generic grammar [1]. A specific instance of this generic grammar provides the communication support for a given 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 cohabit on the same WOS node. WOSP allows the WOS nodes administrators to add and advertise any new service class corresponding to any new specific users' needs without reinstalling WOS. Thus, every WOS node is seen as a set of dynamic service classes created, advertised, updated and deleted on demand. A version of WOSP "spoken" by a WOS node is in fact considered as a local resource.

A WOSP message is a data stream that can be a command or a reply. Details about this syntax can be found in [1]. A simple discovery/location protocol (WOSRP) is used to search for WOS nodes supporting a given version of WOSP. The WOS Request Protocol (WOSRP) serves two purposes : locating versions of service classes (specific WOSP versions) and transmitting WOSP messages to an appropriate server (service class). WOSRP is described in detail in [1].

4. The Resource Discovery WOS Protocol : RD-WOSP

We propose a specific service class (specific version of WOSP) for the configuration and setup of PD applications. This service class is called *Resource Discovery WOSP : RD-WOSP*. We assume that a PD application is composed of a set of heterogeneous software modules. Every module requires a set of resources belonging to one of the two following families :

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

The virtual target machine is a network of WOS nodes ranging from personal devices to regular desktop workstations and to HP parallel computers. The most important service that must be provided by the WOS is the mapping of the PD application onto the target machine. It consists of assigning every module to a WOS node that provides all the resources requested by the given module. In order to guarantee this service, the metacomputing environment should be able to localize the resources requested by the different modules of the PD application.

The services provided by the RD-WOSP service class are :

1. localization of the compute nodes with the appropriate set of resources (discovery service),

2. reservation of the compute nodes taking into account the result of the discovery stage (reservation service) and,

3. setup of the application execution (setup service).

Services can be invoked via the WOS Graphical User Interface or by calling specific routines which have the following syntax : *class-id (service-id, parameters)*, where *class-id* is the class identifier of the specific version of WOSP. The identifier and arguments of the invoked service are respectively *service-id* and *parameters*. In the case of RD-WOSP, these routines are:

1. *RD-WOSP (Discovery, application)* for discovery service. This routine returns the set of WOS nodes providing the resources requested by *application*, where *application* is the identifier of the PD application to configure.

2. *RD-WOSP (Reservation, application)* for reservation service. This routine returns *true* if the reservation is accepted, *false* otherwise.

3. *RD-WOSP (SetUp, module)* for SetUp service. This routine returns *true* if the setup is correctly carried out, *false* otherwise.

4.1. The Discovery Service

This service mainly relies on the WOS environment. It is initiated at the user's request. As a starting point, the user specifies parameters and constraints to execute his PD application, for example, date and time of execution, program to run, execution environment to use, etc. The parameters are translated by WOS in terms of resources.

From that point on, the control is passed onto the WOS to perform the following tasks, if needed:

1. *Locate other WOS nodes*. This occurs only when an insufficient number of WOS nodes that can perform parallel/distributed computing are locally known. A WOS node is assumed to be a candidate for the execution of one or several modules of the PD application, if it implements the RD-WOSP class (*i.e.*, the version of WOSP we are currently describing). The WOS will use the discovery/location protocol (WOSRP) to identify new nodes that understand the version of RD-WOSP.

2. *Locate nodes that can potentially participate in the current request*. In this step, we want to locate nodes (among those identified during step 1) that can be used to answer the user's specific needs and requirements (*i.e.*, nodes that can provide resources requested by the PD application). The request generated by the service requesting machine (machine from which the user is asking the service) must be sent to all WOS nodes collected during step 1.

3. *Collect replies from all the nodes*. Each requested WOS node sends a reply to the requesting WOS node. It will determine which nodes will 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 working configuration.

The search results are conveniently preserved in the local warehouses. Subsequent executions with the same (or similar) parameters may thus reuse the results instead of performing the whole search again.

4.2. The Reservation Service

Here, the WOS simply indicates to the selected node that it will use a certain set of resources, based on the information received. A node can still reject or accept a reservation request.

4.3. The SetUp Service

At a desired time and date, the requesting WOS node sends a command to every node to start the execution. This command is launched by the WOS to setup the reserved resources and initiate the execution of the PD application. The *reservation-no* field indicates the identifier of the reservation request. The WOS just gives the starting signal and waits for the results. It does not interfere with the execution of the application.

5. The High Performance WOSP : HP-WOSP

In our view, a HP application is a sequential or PD application submitted to HP constraints. We assume that HP constraints are temporal constraints imposed by the semantics of the application such as real-time constraints and interactive constraints. HP constraints are usually expressed in terms of CPU performance, bandwidth and time latency of the network. Our goal is to satisfy these constraints by using parallel and distributed computation. In other words, the decomposition of a HP application into a set of communicating modules is carried out only when sequential processing does not meet these constraints. This section describes how the RD-WOSP class is extended to take into consideration the HP constraints of a HP application. Similar to RD-WOSP, the new service class, called *High Performance WOSP (HP-WOSP)*, is composed of three services (routines): *HP-WOSP (Discovery, application)*, *HP-WOSP (Reservation, application)*, *HP-WOSP (SetUp, module)*

Again, *application* is the identifier of the HP application to configure. Reservation and SetUp services of the HP-WOSP class are the same than those of the RD-WOSP class.

In the reminder of this section we will describe how a HP application is represented and how the HP-WOSP discovery service configures these applications.

5.1. The Granularity Tree

A HP application is represented by a tree where each vertex can be recursively decomposed, until all the vertices represent atomic sequential processes. 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, called *Granularity Tree* (GT), shows the degree of granularity afforded by the programmer. The GT does not reveal any information about the precedence rules and the amount of data transferred between processes. For some parallel languages, the GT can be automatically produced, whereas in other cases it may 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. In addition to software and hardware resources, a new family of resources, *HP resource.*, is taken into account by the HP-WOSP class: Some modules may require a minimum threshold of CPU performance or a minimum bandwidth or latency time of the network to be used during communication, etc. The expression of these HP requirements for resources is important in the context of High Performance Computing.

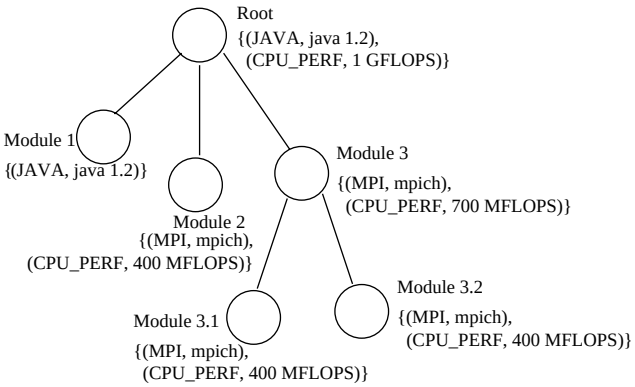

Figure 1. Granularity Tree Structure of a HP application with two levels of decomposition : Root is split into three modules. Module 3 is split into 2 modules

Fig. 1 shows a simple example of a GT. Each vertex is represented by its name and the set of resources it requires. To satisfy the HP constraints, the application must be executed by a WOS node providing at least 100 MFLOPS of effective CPU performance and having a JAVA interpreter. If no node can provide these resources, the application is split into three modules (Module 1, Module 2 and Module 3). If no WOS node can provide resources needed by Module 1 and Module 2, the application cannot be executed. Indeed, the programmer does not propose any decomposition of these modules. In contrast to Module 1 and Module 2, Module 3 can be split into two siblings : Module 3.1 and Module 3.2.

According to this decomposition, the application illustrated in Fig. 1 has three possible versions:

1. Standalone version. The application is executed by one node. In this case the WOS node must provide resources requested by the root.
2. First PD version. Module 1, Module 2 and Module 3 are executed.
3. Second PD version. Module 1, Module 2, Module 3.1 and Module 3.2 are executed.

We should note here that, in a "real" concrete case of HP applications, a manual generation of the Granularity Tree could be a cumbersome task and cannot be applied to global computing applications. One solution of this problem is to develop a specific tool to express and extract, in an homogeneous fashion, the parallelism of a given HP application. This problem is actually being investigated in the framework of another project: ParCeL-2 [4]. More than a new parallel programming language, ParCeL-2 is a new parallel and distributed programming paradigm. Its objective is to provide a minimal set of new concepts to be added to a classical imperative programming language in order to give an "intuitive" expression of parallel distributed applications. ParCeL-2 basically provides a hierarchical syntactic structure that allows the construction of complex processes from elementary processes. The behavior of these complex processes is the result of the parallel execution of the elementary processes they contain.

5.2. The HP-WOSP Discovery Service

The main goal of this service is to locate adequate WOS nodes able to execute a subset of the GT vertices. This subset must however represent the whole application. During the assignment of each vertex, it will be checked whether the selected WOS 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 nodes and the traffic over the network, in order to meet the performances (HP resources) required by the corresponding vertex being assigned. The node selection is made from the top-down. The root node of the granularity tree represents the whole HP application. The HP-WOSP discovery service 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. Obviously, the deeper the vertex in the GT, the higher is the probability to find WOS nodes providing the requested resources and the more fine is the granularity. This recursive process is repeated until each covered vertex is allocated to a WOS node.

The selected size of the grain depends on the characteristics of the parallel machine which will execute the program. Since this machine is a metacomputer, its exact characteristics are only known at execution time. As a consequence, the size of the grain is fixed only at execution time.

6. Conclusion

This paper presents a new approach for configuring and executing High Performance applications on a heteroge-

neous distributed environment. The Web Operating System (WOS) is used as a metacomputing tool supporting the configuration and the launch of the execution of HP applications.

RD-WOSP is in fact used to configure and map Parallel/Distributed applications on a Distributed/Parallel machine. The RD-WOSP service class assumes that communicating modules composing the PD application are known before the execution. HP-WOSP is an extension of RD-WOSP. It allows the configuration of HP applications. Such applications are represented by the Granularity Tree (GT) which indicates the maximum granularity a HP application could exploit during its execution. It also indicates the resources requested by every module composing the HP application. The effective granularity, used during execution and fixed during configuration, is processed in function of the availability and the effective workload of the metacomputer (workload of nodes and network). The decomposition process (parallelization) occurs only if no WOS node can provide the requested resources. This approach breaks with the "classical" alternatives which parallelize a program regardless of the effective workload of the target machine during execution: the granularity is hence fixed *a priori* and therefore, cannot be adjusted during configuration.

At the present time, the code of a given implementation is produced by grouping and "clustering" codes: concurrent execution of several sibling modules on the same WOS node. Nevertheless, for some parallel languages, such as ParCeL-2 [4], it is possible to automatically produce the "optimal" code related to a specific implementation [8]. This optimal code is represented by the serial codes of all GT vertices that will be executed on the same WOS node. In other words, the serial code of a given vertex is automatically extracted from the codes of its sons.

The configuration of a testbed application with a GT of 20 vertices (STORMS:Software Tools for the Optimization of Resources in Mobile Systems) using the HP-WOSP protocol, has demonstrated the feasibility of our solution. Other experiments are in progress to evaluate our approach in large scale HP applications.

From an HPC perspective, HP-WOSP involves many changes in the way of developing HPC applications, which are usually parallel applications. From our viewpoint, these applications should be described in such a way that all implementations could be extracted from the same design (the granularity tree). An implementation is in fact a specialization of a design where all decisions about the specific constraints (requested resources) are made. Furthermore, an implementation should be automatically produced by setting all the constraints. Therefore, we need an *intensional compiler* that can take as input a multidimensional design and all the values of the different constraints. This idea will constitute one of the important perspectives of this work.

Another important aspect of HP applications that this work does not address is the dynamic reconfiguration of the metacomputer when nodes fail or when resources are required for higher priority work. Although not specifically discussed, we think that the approach presented herein offers a strong basis onto which we could develop dynamically reconfigurable metacomputers.

References

[1] G. Babin, H. Coltzau, M. Wulff, and S. Ruel. Application programming interface for WOSRP/WOSP. Technical report DIUL-RT-9901, Université Laval, Québec, Canada, 1999.

[2] S. Ben Lamine, P. Kropf, and J. Plaice. Problems of Computing on the Web. In A. Tentner, editor, *High Performance Computing Symposium 97*, pages 296–301, Atlanta, GA, Apr. 1997. SCS International.

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

[4] P. Cagnard. The parallel cellular programming model. In *8^{th} EUROMICRO Workshop on Parallel and Distributed Processing (EURO-PDP 2000)*, pages 283–289, Rhodes, Greece, Jan. 2000. IEEE Computer Society.

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

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

[7] P. Kropf, H. Unger, and G. Babin. WOS: an Internet Computing Environment. In *Workshop on Ubiquitous Computing*, PACT 2000 (IEEE International Conference on Parallel Architectures and Compilation Techniques), pages 14–22, Philadelphia, PA, 2000.

[8] P. Kuonen, G. Babin, N. Abdennadher, and P. Cagnard. Intensional high performance computing. In P. Kropf, H. Unger, G. Babin, and J. Plaice, editors, *Distributed Communities on the Web (DCW 2000)*, Lecture Notes in Computer Science, Quebec, Canada, June 2000. Springer Verlag.

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

[10] 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.

[11] H. Unger. Distributed Resource Location Management in the Web Operating System. In A. Tentner, editor, *High Performance Computing 2000 (ASTC)*, pages 213–218, Washington, DC, 2000. SCS International.

[12] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. WebOS: Operating system services for wide area applications. In *7th IEEE Symposium on High Performance Distributed Systems*, Chicago, IL, July 1998.