A Resource Management Tool for Heterogeneous Networks

Andrea Clematis
IMA - CNR
Via De Marini, 6
16149 Genova (Italy)
clematis@ima.ge.cnr.it

Gabriella Dodero, Vittoria Gianuzzi
DISI - Università
Via Dodecaneso, 35
16146 Genova (Italy)
{dodero,gianuzzi}@disi.unige.it

Abstract

We describe the design principles and implementation of a tool to be used as Resource Manager on arbitrary networks of workstations. It evaluates both statically (off-line) and dynamically (on-line) the computational power and workload of each node in the network, in order to select the most performant computers after each application request for task spawning to the network. The tool is a component of a system to implement Parallel Virtual Libraries on heterogeneous networks of workstations. [1]

1. Introduction

Technology evolution together with market pressure has made it possible to exploit networks of workstations as efficient supercomputing tools. The high speed networks connecting increasingly faster systems may now substitute expensive parallel supercomputers at a fraction of their cost. Given the non-homogeneity usually present in this kind of nets, an efficient distributed programming environment with a specialized competence in scaling and locating parallel applications, able to maintain load balancing among heterogeneous workstations, is needed, to obtain the maximum efficiency.

Various research fields rely on efficient solutions for computationally heavy problems. Such applications are often made portable by using mathematical software parallel libraries, which are optimized for the underlying architecture. Many such libraries have been written specifically for parallel supercomputers. As of today, a similar effort has not yet taken place for networks of workstations: there are few environments that support an easy–to–use access to heterogeneous distributed systems, where mathematical libraries are available, with transparent access to resources. Usually, either they have a complex user interface, or they have limitations in architectural support. The goal of providing and supporting a *Parallel Virtual Library* for arbitrary networks of workstations, including the heterogeneous ones, is still far from being achieved. This research area is thus a most promising one where novel systems are now being proposed.

It is important to identify the hardware/software environment in which to operate, the features of the parallel applications to be managed, and the resource manager system properties which are desirable, before selecting a specific design.

As for hardware/software environment we consider LAN of workstations, as usually available in R&D laboratories. They are composed by a few tens of hosts, with different degrees of homogeneity. As discussed in [1], three kinds of heterogeneity can be considered, that is: *Configurational* (hosts have the same architecture but different configuration), *Architectural* (executable files cannot be exchanged), and *Operating System* heterogeneity. Completely homogeneous systems are seldom found, but at their initial purchase. For example, a Pentium-based laboratory may soon become heterogeneous, due to CPU update, memory/ peripheral extensions and so on. Two laboratories have been considered, the one of the IMA-CNR and the student lab of DISI. The first one is highly heterogeneous, both in configuration and architecture, with different Unix-like operating systems. The second one is composed by about 45 Pentiums, from 133 to 350 Mhz, with different configurations, all of them running Linux.

The kind of applications considered are those using parallel libraries, each function of which is implemented using SPMD or master/slave paradigm. That is, each process (except for the master) executes the same code, and exhibits a similar computational/communication profile. Thus, our resource management system need not be a general purpose one.

In order to identify which computational resources are to be managed, we first considered the possibility of having

[1] This work has been partially supported by a grant of Italian CNR (Virtual Libraries for Computational Problem Solution).

the exclusive use of the network (e.g. night hours or weekends). However, this choice resulted unacceptable: owners of workstations dedicated to research projects (e.g. powerful Indigo for graphical applications) were willing to give access to others during idle periods, but would be reluctant to establish in advance time periods during the week when their workstations would be available to others. As a consequence, we decided to design a system entirely working at user level, which does not require the exclusive use of the network, "light" with respect to execution time overhead, installation difficulties, and ease of use.

The system we designed, with the support of a CNR grant, has been named PINCO [2](Programming envIronment for Network of COmputers); it provides a safe user interface, that activates the required parallel functions on the workstation cluster which, at each moment, may guarantee the best performance. Such a tool is then essential to efficiently implement a Parallel Virtual Library.

In the following Section we shortly outline similarities and differences to other resource management systems. Then, we describe PINCO architecture its structure and how load data information is considered. Tables will be shown, related to benchmark results obtained on a highly heterogeneous network. Two examples of parallel processes allocation are then presented.

2. Related Works

The idea of remotely executing library functions is implemented by NetSolve [2], a net server for the solution of computational problems developed by the Jack Dongarra group. The main features of NetSolve are the availability of different user interfaces, either interactive (graphics) or Fortran and C embedded, and the ability of managing computational resource requests on a distributed system, taking into account load balancing and fault tolerance. Presently, however, no special support is provided for parallel library management.

Besides, there exist other industrial resource-management tools which offer several services for heterogeneous workstation networks. Well known are CODINE [3], by GENIAS, and Load Sharing Facilities (LSF) [4], by Platform Computer. They support facilities for batch queing, job management, deadline scheduling, in a word, the maximum utilization of the resources, at day time and overnight. However, their aim is to reach the net maximum efficiency, rather than offering a specific support for parallel programming, which is possible, but not privileged.

A more complex system (described in [7]) has been implemented in the framework of the Esprit project EROPPA, for the execution of computationally intensive parallel 3D-rendering jobs. Here, the emphasis is given to management of the large volume of data these applications require or generate. CODINE has been selected for the job management layer.

Finally, we recall UTOPIA [1], a load sharing facility implemented on top of Unix for very large and heterogeneous systems. As well as LSF it is a general purpose, user tranparent system, which supports remote execution only at task initiation time. It is a more powerful tool than PINCO, which is on the contrary devoted to manage distributed and parallel applications.

DAME [5, 6], is, from certain points of view, the system closest to PINCO, since it aims to dynamically balance the workload of SPMD regular computations. The most significant dfference is that the reconfiguration protocol is based on data migration instead of task migration, thus requiring the cooperation of the application programmer, which must implement decomposition-independent programs.

3. PINCO Architecture

PINCO has been designed as composed by three parts: the *Job Scheduler* and the *Resource and Task Manager*, which provide the related facilities, and the *Application Interface* which acts as intermediate level between the application and the other two components. Moreover, a programming environment is provided, to help PINCO programmers in developing distributed applications. Keeping system components as much separate as possible is useful to experience on different job and resource management techniques, by substituting individual components with others implementing alternative policies.

Each parallel job is executed on a partition, that is, on a subset of nodes. Due to the hetereogenity of the hardware architecture, some form of normalization is needed in order to dynamically evaluate and compare the computational power of each node. The unit we choose is the *tenth of a Pentium 133* (ETP). That is, at each moment, PINCO assumes to have a virtual pool of Pentium 133 unloaded, and satisfies each request selecting a set of physical machines which offer the greatest number of ETP (to minimize the internode communications). PINCO partition choice will be *adaptive*, that is, the partition will be automatically defined by the scheduler, considering system availability and user requests.

So far we have not yet taken into account the performance of the underlying network, either as absolute or relative performance (with respect to some conventional unit of traffic). The reason is due to the relatively low impact that such figures may have on our intended applications, which are computationally intensive and less sensitive to traffic congestions.

[2] 'pinco' was also the name of a kind of traditional Ligurian sail ships used during the 19^{th} century for commerce in the Mediterranean

Presently, only a part of PINCO has been implemented, that is:

PPE the PINCO Programming Environment which allows to compile and to allocate both PINCO and the parallel application, and which executes the off-line evaluation of the computing power for each kind of network of workstations (see Section 4);

PRTM the distributed PINCO Resource and Task Manager, which periodically evaluates the workload of each network component, processes programmer requests for the computing power needed by the applications, and eventually activates the parallel tasks (see Section 5).

In case of significant workload changes on some node of the network, as in practice is often the case, provisions for tasks migration must be taken. Reconfiguration has not yet been implemented, even if tools supporting task synchronization and process migration are already allowable. [8, 9].

In order to start experiencing with distributed resource allocation, since the scheduler is yet under way, a simple API has been implemented, by means of a limited set of calls to spawn, control and monitor tasks under the control of PRTM. It demands to the programmer the burden of deciding about task allocation: in such a way it is possible to easily experience with different scheduling policies, before embedding them in the system. In other words, the allocation policy is *variable* [10], since the partition is determined at job submission time, only considering user requests.

Thus, the already implemented part of PINCO is available to statically allocate parallel programs, to provide the suitable computing power and to keep a balanced load on the net.

To achieve maximum system portability, an already available platform must be selected. The software platforms we considered were Paradise, an extension of Linda [11] and PVM [12]. Paradise is an environment for parallel and distributed computing, based on the model of virtual shared memory, while PVM (Parallel Virtual Machine) is a communication library for distributed memory machines, which is becoming the most widely used one on networks of workstations. PVM is also upwards compatible with MPI, the standard for distributed applications, which as present is still not so widely available.

The present version of PINCO is implemented on top of PVM, and its architectural model is similar to that of PVM, that is, it is based on the *PINCO Master Daemon*, running on the master host, usually, the most performant of the net (see Figure 1).

The master daemon is a process, unique in the network, composed by three software modules: the *Load Monitor* (LM), the *Resource Manager* (RM) and the *Communication Library INterface* (CLIN).

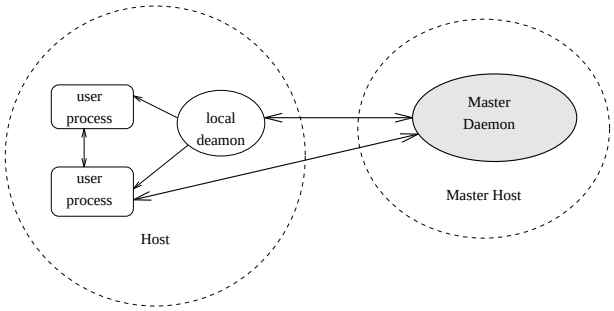

Figure 1. Daemons relationships

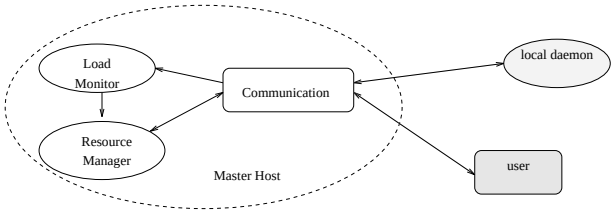

Figure 2. PINCO Master Daemon structure

CLIN includes all the PVM based communication functions: porting of PINCO to other parallel or distributed machines, with different communication features (like Paradise, or MPI-2) is made easy by such encapsulation. **LM** collects the load situation of each host, while **RM** is responsible for balancing the workload among multiple execution hosts. **RM** also acts as *Submission Manager*: the application sends computing power and task activation requests, obtains the service and a report with other useful information. When migration will be included in PINCO, it will be supported by RM as well.

On each host, the PINCO Local Daemon is executed. It is composed by three modules: the *Local Load Monitor*, the *Process Handler* and the *Local CLIN*. The first one evaluates the local load, which is periodically sent to the Master Daemon. The Process Handler is responsible for activating local tasks, according to the Master Daemon requests, and for sending back information about status of the computation, when needed.

The choice of building PINCO on top of an already existing communication library also allows to easily obtain additional and useful features such as signal handling and terminal I/O for each remotely executed process.

Remark that application processes may communicate among themselves using any library, i.e. Unix sockets, PVM, Linda or others, since PINCO does not interfere with application communications.

4. PINCO Parallel Environment

Considering that PINCO run time environment is made by heterogeneous nodes and the code distribution is decided in a dynamic way, it is necessary to generate code for each possible different target architecture. The word architecture is used in this case as a synonym of operating system version and it identifies executable code compatibility.

Other parallel libraries like PVM, MPI, Linda, take in consideration the possibility of generating code for different target architectures, and this is one of the key points in providing PINCO portability. Most of the work is however left to the user who is charged of the task of recompiling code for each different target architecture he/she wants to include in the parallel machine, and then of placing each copy of executable code in the proper directory. This may become a tedious task at least when the number of heterogeneous architectures is greater than two.

4.1. Compilation environment

PINCO provides a simple but effective compiling system which is able to automatically generate code for any set of target architectures. The only assumption is that a common file system can be accessed by all nodes in the network. Then, the user has only to set two parameters:

⋆ a variable which indicates the target application, this is represented by the PINCO_ROOT environment variable in Unix systems, and normally indicates the root directory containing the source code of the application;

⋆ a configuration file which contains the names of selected target architectures.

As a result the PINCO compilation system generates code for the different target architectures and distributes it so that it will be accessible from any machine of the parallel system.

The final file system configuration for the PINCO environment is represented as follows:

```
$PINCO_ROOT
/ (PINCO utilities and conf. files)
   SRC (contains the source code)
   INCLUDE (header files)
   OBJ (object files  sorted by archit.)
   BIN (binary im-
ages sorted by archit.)
   DAT (application data)
   LIB (libraries sorted by archit.)
```

This logical organization may be mapped in different ways on the physical file system, depending on network organisation.

4.2. Evaluating System Resources

For an efficient management of computing resources available on the local area network, PINCO has to be able to evaluate resources, and this a challenging task since we are considering an heterogeneous and shared environment, and these characteristics have to be handled carefully.

A first point is that of evaluating the total computing power of the parallel system as the sum of heterogeneous resources. This point could be very subtle for different reasons like the fact that different machines may be more or less suited for different types of applications (e.g. matrix multiplication and sorting algorithms), and many factors may determine the actual efficiency of a single node in processing a selected application (CPU speed, memory hierarchies, specialised coprocessors and the like).

PINCO approach to this problem has been taken considering the present target network and a set of suitable benchmarks: Table 1 summarises the set of available machines, while Table 2 summarises the set of considered benchmarks.

NAME	MODEL	OP SYS
paperoga	Sparc Station 20	OS4.1.3_U1
athena	Sparc Station 20	Solaris 2.6
helios	Sparc Station 1	OS4.1.3
aphrodite	Sparc Station 10	OS4.1.3
elba	Sparc St. Classic 4/15	Solaris 2.5
fred	HP 9000/715	HP-UX 9.01
venus	Sparc St. Classic	Solaris 2.5
paperinik	Sparc St. Classic 10	OS4.1.3_U1
bergeggi	Sparc Station 5	Solaris 2.5
asterix	UltraSparc 5	Solaris 2.6
gauguin	Indigo 2	IRIX 6.2
matisse	SGI O2	IRIX 6.3
baobab	Pentium 133 Mhz	Linux

Table 1. IMA-CNR local area network

Running the benchmarks on the different nodes, it is possible to get performance figures for the whole network. These figures are not strictly defined because the different nodes may change their relative performance selecting a different benchmark. In any case it is possible to get an approximate value of relative speeds of the different machines, which may be used in most cases as suitable indication. Table 3 summarizes the results for two different benchmarks, running on different workstations of the network. The first column lists node names, which have been described in Table 1. The second and third columns respectively show the floating point and fixed point performance of these nodes at a given network workload. Nodes are ordered for floating

NAME	DESCRIPTION
sim	An integer program that compares DNA segments for similarity.
fhourstones	An integer program to find solutions to the 'connect-4' game
dhrystone	A synthetic integer program which provides a MIPS rating based upon a 'typical' instruction mix. Very sensitive to optimization: results will appear erratic unless degree of optimization is carefully tracked.
nsieve	An integer program which generates prime numbers based on the Sieve of Eratosthenes using array sizes from 8 KBytes to 2 MBytes. A nsieve MIPS rating is provided but the nsieve instruction mix is NOT 'typical'.
heapsort	An integer program which uses the 'heap sort' method of sorting a random array of long integers up to 2 MBytes in size. A heapsort MIPS rating is provided but it is NOT representative of a 'typical' instruction mix.
hanoi	An integer program which solves the Towers of Hanoi puzzle using recursive function calls
queens	An integer program that solves the "14 Queens Problem". That is, the program finds all ways to place 14 queens on a 14 X 14 square "chess" board such that no queen can capture another.
flops	Estimates MFLOPS rating for specific FADD, FSUB, FMUL, and FDIV instruction mixes. Four distinct MFLOPS ratings are provided based on the FDIV weightings from 25% to 0% and using register-register operations. Works with both scalar and vector machines. Since the program tries to maximize register usage the results are NOT sensitive to main memory speed. In this sense flops yields a peak rating.
clinpack	This is the Linpack program (floating-point) converted to C. Results here are sensitive to cache effects and memory speed.
fft	An FFT test program from Ron Mayer (mayer@acuson.com). Program times a 131072 point FFT.
hintHINT	measures "QUIPS" (QUality Improvement Per Second) as a function of time. "Net QUIPS" is a single number that summarizes the QUIPS over time.

Table 2. The set of benchmarks

point decreasing performance, the fixed point performance being quite close to it.

NAME	KFLOPS (clinpack_dpu)	SEC (queens)
gauguin	78882	24.02
asterix	28027	34.05
matisse	21313	36.06
baobab	10917	35.01
bergeggi	10382	55.58
paperoga	8770	81.97
athena	8354	83.19
paperinik	7289	103.98
aphrodite	5955	126.20
helios	1213	464.52

Table 3. Two benchmarks

We identify with s_i the relative speed of the i-th machine for $i = 1..N$, where N is the total number of nodes. Values s_i are obtained by selecting a reference machine j which will have $s_j = 1$ and then defining the other s_i comparing the performance of each machine with that of machine j. In this way we will obtain the total available power as the sum of all s_i. In general this value is different from N. This provides a static approximation of the computing power of the unloaded system. The reference machine we took is the node baobab, a Pentium 133 with Linux.

PINCO is designed to run on shared systems, hence we have to consider that actual computing power of each node will be only a fraction of the computing power of the same node. This fact can be expressed by the following simple formula: $ds_i = (1 - \sigma_i)s_i$, which correlates the dynamic relative speed with the static relative speed of node i. Here the parameter σ_i expresses the load factor of node i. In Unix based systems, a simple but effective way of evaluating load

factors is to use the percentage of CPU usage as provided by ”ps” system call. Actually this is the choice currently implemented by PINCO.

5. PINCO Resource and Task Manager

PINCO architecture described in Section 2 has evidenced that there does not exist a single component which may be called ”the Allocator”. Such a functionality is distributed across the network, yet maintaining a centralized information collector, that is the Resource and Task Manager. The latter is a process running on the host node, where information sent by the various instances of the Load Monitor is kept, and used to take decisions about process allocation to the network.

As we shall see, at present allocation decisions result from a negotiation process with the user process, as the choice suggested by the PRTM is evaluated and partially modified, if needed, by the user. As far as experience in this decision making process is collected, it is expected to include ”smart” allocation policies in PRTM, thus making user negotiation no longer needed for most applications.

Information to be collected by the Load Monitor can be classified as:
⋆ static info: initialized as collected from off-line benchmarks (see Sect. 4.2);
⋆ dynamic info: computed and periodically updated by the various local daemons.

Dynamic information is computed by each Local Load Monitor, running on a different PINCO host computer. The percentage of CPU free is computed every 5 seconds by issuing a local system call, such as the command ”ps” from a shell. This value is sent again only if different for more than 10% from the one previously sent to the Load Monitor.

When the Load Monitor receives a message from the Local Load Monitor at node **x**, it updates the so called *load_table* where dynamic and static network load parameters are kept. The static ”weight” of node **x** is multiplied by the percentage of CPU free, thus giving the available power on node **x** for PINCO requests. This value is still expressed in ETP.

5.1. Resource management

Allocation requests from user processes are based on a data structure called *grant_record*. It is filled by PRTM and passed to the user process by means of a two phase protocol:

1. in the first phase, the user process issues a request expressed by the call
   ERR=PINCO_GRANT (REQUESTED POWER, GRANT_RECORD);
2. the user process may check compliance of the proposed allocation to possible additional constraints (see in the examples one such instance) and then issues the activation request:
   ERR=PINCO_SPAWN (EXECUTABLE FILE, GRANT_RECORD);

The function PINCO_grant fills the following data structure, in accordance with values actually stored in the *load_table*:

```
struct grant_record
    {    float grant_power;      /* granted power */
         int cluster_dim;        /* host number */
         node * topology;
    }    grant_record;
```

where node is a list of cluster_dim pairs (processor_name, local_granted_power), such that the sum of all local_granted_power fields, as stored in grant_power, matches the request or default approximates it if it is impossible.

Loads corresponding to local_granted_power are reserved for the process that invoked the PINCO_grant, but parallel processes are not immediately spawned. This happens only when a PINCO_spawn is invoked by the user process, where the actual number of parallel tasks per processor is specified, by adding a third value in the node list of the grant_record. In case the user process does not issue a PINCO_spawn within a predefined delay, reserved resources are however freed. The user process may also update the grant_record, before invoking a PINCO_spawn, in order to use just a subset of available resources, e.g. by not using certain nodes, or leaving some fraction of computing power to other applications.

Example 1

Let us assume that we wish to multiply 256*256 matrices and that we estimate a power of 16 Pentium-units for such job. A request is issued; the grant_record contains the following node list:

```
asterix      3.5
gauguin      8.5
bergeggi     3.5
helios       0.5
```

which satisfies the computing power requirements. The user process then decides to spawn 32 processes, each one handling 8 data rows, and to allocate them so that each one may take advantage of 0.5 Pentium-units of computing power. The node list to be passed to PINCO_spawn shall be:

```
asterix      3.5     7
gauguin      8.5    17
```

```
bergeggi     3.5     7
helios       0.5     1
```

Example 2

Let us assume that the user process is efficiently parallelizable only by creating a number of tasks which is a power of 2. A request for 16 power units is issued: The grant_record contains the following node list:

```
asterix     3.5
gauguin     3.5
bergeggi    3.5
helios      0.5
```

Since granted power is below the requested power, the algorithm must be resized to 8 tasks only, which can be allocated to the first three processors only, neglecting the already overloaded fourth node, helios. Each task may use one Pentium-unit, as the following node list shows, passed to PINCO_spawn:

```
asterix     3.      3
gauguin     3.      3
bergeggi    2.      2
```

Remark that the latter is a sublist of the former, and that there is still some computational power available at each node.

6. Conclusions

We have described in this paper the design principles and the architecture of a toolset aimed at supporting the development of Parallel Virtual Libraries on heterogeneous networks of workstations. Two components of such toolset already exist, which together may act as a resource manager. A simplified API is provided, in order to collect scheduling experiences before actually implementing a Job Scheduler. Examples have been shown of how computing power is statically and dynamically evaluated on the network, and of how process spawning is negotiated between the Resource Manager and the application.

The subset of PINCO which has been described in the paper is now in the process of extensive testing; at present we are evaluating its performance in terms of overhead with respect to PVM and its additional possibilities for tuning parallel library routines to heterogeneous networks.

References

[1] S. Zhou, J. Wang, X. Zheng, P. Delisle. UTOPIA: a load sharing facility for large, heterogeneous distributed computer systems. *Software - Practice and Experience*, 23:1305-1336, 1993.

[2] http://www2.netsolve.net/

[3] http://summa.physik.hu-berlin.de/rosen/hfill codine-eng.html

[4] http://www.science-computing.de/produkte/lsf/

[5] http://russell.ce.utovrm.it/dame/

[6] M.Cermele, M.Colajanni, G.Necci. Dynamic load baloancing of distributed SPMD computations with explicit message-passing. *Proc 6th Heterogeneous Computing Workshop*, 2-16, IEEE Computer Society Press, 1997.

[7] A.N. Meliones, I. Barosan, T.T. Varvarigou. Efficient distributed execution of computationally intensive tasks. *Proc. HPCN'98*, LNCS 1401:262-273, 1998.

[8] A.Clematis, G.Deconinck, V.Gianuzzi. A flexible state-saving library for message-passing systems. *Proc. 6^{th} Euromicro Workshop on Parallel and Distributed Processing*, IEEE Comp. Soc. Press, London, 1998.

[9] J. Casas, R. Konoru, S.W. Otto, R. Prouty, J. Walpole. Adaptive load migration systems for PVM. *Proc. Supercomputing'94*, 390-399, 1994.

[10] D.G.Feitelson, L.Rudolph, U. Schwiegelshohn, K.C.Sevcik, P.Wong. Theory and practice in parallel job scheduling. *IPPS'97 Workshop Job Scheduling Strategies for Parallel Processing*, 1291:1-34, 1997.

[11] N.Carrero, D.Gelernter. Linda in context. *Communication of the ACM*, 32(4):444-458, 1989.

[12] V.S. Sunderam, G.A. Geist, J. Dongarra, R. Manchek. The PVM concurrent system: evolution, experiences, and trends. *Parallel Computing*, 20(4):531-546, 1994.