Breaking the Curse of Dynamics by Task Migration: Pilot Experiments in the Polder Metacomputer

B. J. Overeinder and P. M. A. Sloot

University of Amsterdam, Department of Computer Science
Parallel Scientific Computing & Simulation Group
Kruislaan 403, NL–1098 SJ, Amsterdam, The Netherlands
`e-mail: {bjo,peterslo}@wins.uva.nl`

Abstract. With the advent of high speed networks, distributed cluster computing and metacomputing have assumed an enormous interest. However, software methods and techniques to make the full potential of these distributed environments available, are not yet mature. In this paper, we focus on dynamic load balancing of resources and applications as one of the crucial techniques to optimize performance in distributed environments. Some design and implementation details are described, and early experimental results are presented.

1 Introduction

The current developments in clusters of workstations, and on a larger scale metacomputing environments, indicate that resource management has become the instance that determines the efficacy of the distributed computing environment. In distributed environments the typical set of jobs consists of interactive and batch jobs, which in turn can be sequential or parallel execution runs. By the diversity of the jobs offered to the distributed environment—interactive users start sequential and parallel jobs, and batch jobs arrive with some arrival probability distribution function—the demand for and the availability of resources is highly dynamic.

Resource management in distributed environments spans a variety of activities such as job scheduling, I/O scheduling, load balancing, etc. In order to optimize performance of applications or the utilization of resources, the resource management system should be able to react on changes in the distributed computing environment. As a consequence, several provisions have to be made available to the metacomputer in order to interact with resources and applications.

A serious problem hampering the development of metacomputing environments is the lack of a sound theoretical basis for resource management strategies to build upon. In order to break the impasse, we develop an experimental environment that provides a framework for the development and evaluation of the various components making up the metacomputer. The experimental environment is essentially a metacomputer in its functionality and characteristics, but allows to study, for example, different policies for resource management or test designs and implementations of scalable I/O libraries, and the validation of theories.

In this paper we concentrate on issues concerned with dynamic load balancing of parallel applications in a message passing metacomputing environment. This paper is outlined as follows. Section 2 describes the current hardware and software trends in metacomputing. In Section 3 the Polder metacomputer framework is introduced and Section 4 presents the design and implementation considerations of the load balancing facility incorporated with a message passing library. Experiments and results are presented in Section 5. The results of the experiments are discussed and summarized in Section 6.

2 Background

The current developments in high performance cluster computing and metacomputing are moving along two axis: hardware and software. The hardware development of parallel supercomputing and modern networks/clusters of workstations are directing to the same vanishing-point on the horizon. The compute nodes in the parallel supercomputer are the same processors found in workstations, and the performance of the distinguished proprietary interconnection networks are attained by independent available network interfaces such as Fibre Channel, HIPPI, or Myrinet. Progress in wide area networking, e.g., ATM, motivated the development of software infrastructures that smoothly integrate distant distributed resources into a metacomputer that enables the coordinated implementation of high performance applications.

2.1 Trends in Hardware

The development of high speed networks, both for local area and wide area networks, has triggered a refocus on the hardware used in high performance computing, and in particular a refocus on distributed memory architectures. For example, the Massively Parallel Processors (MPPs) that are used to solve large computational problems, are distinct by their proprietary message passing networks, i.e., communication backplanes specifically designed for a family of MPPs. With the advent of fast network interfaces that are generally available, like FDDI, (switched) Fast Ethernet, Fibre Channel, HIPPI, and Myrinet, the same large computational problems can be solved effectively on clusters of workstations connected by a local area network (LAN). In particular Myrinet is an outstanding example of how technology used for communication and switching in MPPs has evolved to a high speed LAN.

The availability of high speed LAN has initiated a number of research projects to build parallel supercomputers made of Commodity Off The Shelf (COTS) components. Although the projects described below also cover software issues, their main focus is the implementation of a parallel supercomputer.

The Beowulf project [13] aims to develop a parallel computer architecture based upon Pentium Pro processors and switched Fast Ethernet communication links (i.e, switched Fast Ethernet is not used as a broadcast medium, but rather as a point-to-point interconnection fabric giving the full 100 Mbit/s bandwidth). In addition with the availability of powerful, free operating systems (Linux, FreeBSD) and message passing interfaces (MPI), the Beowulf project realized a low-cost commodity parallel computer. With a 16 node parallel computer a sustained performance of one Gflop/s has been obtained on scientific applications.

An interesting initiative that combines both high speed LAN and WAN interconnections in the implementation of a high performance computing platform is the Distributed

ASCI Supercomputer (DAS) [4]. (Note that ASCI stands for Advanced School for Computing and Imaging—a Dutch research school.) The DAS is a 136-node wide-area distributed system built out of four Myrinet-based Pentium Pro clusters. The four clusters are located at four universities: Free University Amsterdam, University of Amsterdam, Delft University of Technology, and University of Leiden.

Each node contains a Pentium Pro, 64 MB RAM, a 2.5 GB local disk, a Myrinet interface card, and a Fast Ethernet interface card. The nodes within a local cluster are connected by a Myrinet SAN network (SAN stands for system area network), which is used as a high speed interconnection, mapped in user-space. Fast Ethernet is used as the operating system network for NFS services, etc. The four local clusters will be connected by an ATM wide area network, so the entire system can be used as a 136-node wide-area distributed cluster (see Fig. 1). The system runs the BSD/OS (version 3.0) operating system from BSDI.

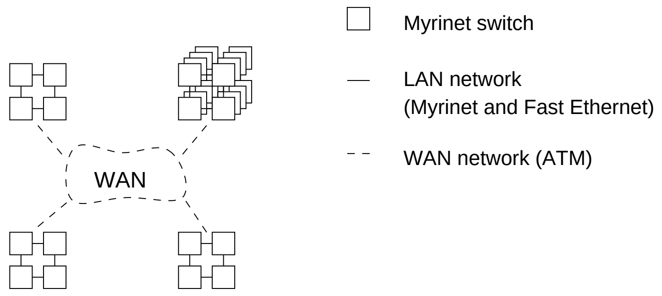


Fig. 1. Overview of the DAS Architecture. Four local area Myrinet clusters are connected by an ATM wide area network.

The DAS distributed supercomputer, with its high speed local area and wide area interconnections, can be regarded as a prototypical metacomputer architecture for the near future. In this respect, the DAS architecture provides a unique experimental testbed for research in metacomputer software infrastructures.

2.2 Trends in Software

New technologies in wide area networks has resulted in a new impetus to research directed to provide coordinated network services. The feasibility of wide area high speed network technology (e.g., ATM, but also HIPPI on SONET) has been demonstrated by the implementation of network testbeds including BERKOM, CASA, and I-WAY. The aggregation of distributed and high performance resources on high speed networks will change the perspective on distributed computing and have an impact on the development of scientific applications. In a similar way as parallel computing enabled scientists to solve computational problems that could not be obtained efficiently by sequential computing, aggregated distributed resources can engage larger computational power to a single application.

Although the hardware developments in high speed networks are impressive, the services provided to use the aggregated distributed resources in a coordinated manner are still in their infancy. To fully exploit the potential of distributed resources on coordinated networks, a software infrastructure must be developed to provide an easy to use and transparent access to the resources. This software infrastructure, the *metacomputer*, manages the complexity of the underlying physical system for the user. The key observation in metacomputing environments is that with the current conceptual model interacting autonomous hosts are stretched into a regime for which it was not designed. This has resulted in a collection of partial solutions without coherence and scalability. The challenge is to provide an integrated foundation that hides the underlying physical infrastructure from users and from the majority of programmers. By smoothly integrating the diverse computational resources, the metacomputer provides a platform that fulfills the requirements of a new class of resource-intensive applications.

Two projects that are exemplary for the current trends in metacomputing research are Legion and Globus. A prototype of the Legion metacomputer and preliminary versions of Globus components have been demonstrated successfully as part of the I-WAY network experiment [6].

Legion is a metacomputer project designed to provide users with a transparent interface to the available resources, both at the programming interface level as well as at the user level [8]. Legion uses an object-oriented framework that enables a coherent solution to problems like support access, location, fault transparency, inter-operability, security, etc. The objects, written in either an object-oriented language or other languages such as C, will interact with other objects via well-defined interfaces. The use of objects allows for substantial flexibility in the semantics of user applications; a user is able to select both the kind and level of functionality, and make their own trade-offs between function and cost (e.g., the level of security in authentication).

The Globus [7] project addresses the metacomputing challenge by a vertically integrated treatment of application, middleware, and network. In the Globus perspective, metacomputing can build on distributed and parallel software technologies, but also requires significant advances in mechanisms, techniques, and tools. The metacomputing software problem is approached from the bottom up, by developing basic mechanisms such as communication, authentication, network information and data access. These low-level components define a metacomputing abstract machine on which can be constructed a range of alternative infrastructures, higher-level services, and applications.

The long term goal of the Globus project is to construct an integrated set of higher-level services that enable applications to adapt to heterogeneous and dynamically changing metacomputer environments. The adaptive applications are able to configure themselves to fit the execution environment and optimize the performance.

Essential to the success of metacomputing is careful scheduling. Generally, there are two performance optimization objectives in wide-area systems: high performance computing (reducing turnaround time of jobs) and high throughput computing (e.g., maximize the aggregate amount of work per time period). Given one of these two goals, the scheduling process must decide where a job and its constituent tasks will run. The objectives and issues that must be addressed by a wide-area scheduling system are more complex than in local cluster scheduling systems [14]. For example, the wide-area scheduler should make use of the heterogeneity in the metacomputer by efficiently exploiting remote resources. However, in a metacomputing setting, resources are often managed by separate local schedulers (e.g., Condor, Codine, LSF) which are not coordinated. Consequently, the wide-area scheduler must make decisions in concert with the local site schedulers.

The delicate interplay of the wide-area scheduler with the local site schedulers is one of the research interests in the Polder metacomputer project, which is presented in the next section.

3 The Polder Metacomputer Experimental Framework

The Polder metacomputer initiative [11] is an ambitions project that aims to provide an experimental framework for metacomputer design tradeoffs and gradually build a metacomputer environment that organizes heterogeneous distributed resources into one single computing environment with a unified access. By its distributed nature, the resources are administered by local authorizing resource managers. Therefore, the Polder metacomputer must incorporate existing management software concerning resource control, access control, accounting and monitoring while supporting the multitude of hardware platforms present within the distributed system.

In the Polder metacomputer experiment different ways of use of metacomputing are addressed: high performance computing, high throughput computing, multi-site computing and automatic task balancing for dynamic resources. Each of these different usages of the metacomputing environment has its own requirements with respect to the services provided by the metacomputer. The underlying mechanisms should be flexible and generic in order to efficiently support these different requirements in services. To tackle these problems, a number of subprojects have been initiated to deal with issues like metacomputer access and job submission, wide-area and local scheduling, load balancing, and scalable I/O. These subprojects are performed by the different participants in the Polder initiative, among which the University of Amsterdam, NIKHEF (Amsterdam), Delft University of Technology, University of Wisconsin–Madison, and Paderborn Center for Parallel Computing.

Some of the issues concerning metacomputer access and job submission, scheduling, and load balancing are discussed in the next sections. Within the MOL partner project, the PLUS lightweight communication interface [2] addresses inter-operability between heterogeneous platforms and different message passing layers. PLUS encapsulates message passing specific communication primitives (e.g., `MPI_Send`, `pvm_send`) and enables inter-operability between MPI and PVM applications.

3.1 Resource Management in the Polder Metacomputer

The global resource management structure of the Polder metacomputer model is depicted in Fig. 2. The structure determines how the heterogeneous distributed resources are presented to the metacomputer user or application. On the base-level there are resources (e.g., workstations, MPPs, or I/O devices) administered by a local resource manager (e.g., Condor, Codine, or LSF). The aggregated local resources (i.e., at the base-level the resources administered by the local resource manager) are represented by self-describing active agents. These agents (in Fig. 2 the entities in the shaded area) describe the type of resources, amount of memory, disk space, connectivity, etc.—the agent is essentially not limited in its descriptive plurality. The agents can be aggregated into a new agent, and hence represent a larger set of distributed resources. The aggregation of agents and the information advertised by the agents can reflect local authorization decisions. Although the organization of the agents is hierarchical, the perspective to resources is one-dimensional; that is, a unified view to the heterogeneous distributed resources on a coordinated network.

The Polder metacomputer access interface is distributed and WWW-based to allow for a scalable, flexible and generic interface that interacts with the resource agents. The wide-area resource management actually takes place at the metacomputer access interface. Upon job submission via the access interface—with the job requirements being specified—the agents start with bidding on the job. According the wide-area scheduling policy, one of the agents offers the best fit on the job requirements. The job and its constituent tasks are allocated to the resources in coordination with the local resource manager. In this top-down approach, the wide-area scheduler determines the resources assigned to a job, and direct the local resource managers to actually allocate these resources.

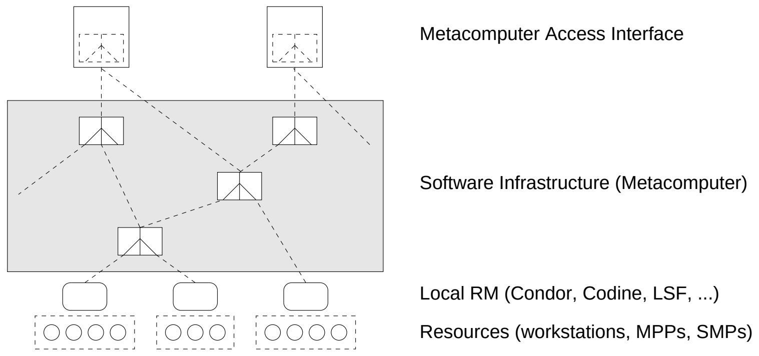

Metacomputer Access Interface
Software Infrastructure (Metacomputer)
Local RM (Condor, Codine, LSF, ...)
Resources (workstations, MPPs, SMPs)

Fig. 2. The Polder metacomputer global resource management model. The software infrastructure (active agents) organizes the distributed resources to a metacomputer.

Wide-area scheduling is a complex problem and subject of various research projects. Within the Polder metacomputer project a simulation model of the resource management infrastructure has been developed to allow for rapid prototyping and evaluation of scheduling strategies. Experiments with scheduling strategies under strict conditions can be instrumented on top of the resource management simulation model, which is essential for validation with theoretical models. After a scheduling algorithm has been thoroughly evaluated, it can be integrated within the metacomputing environment.

3.2 The Curse of Dynamics

In general the resources in the metacomputing environment are not exclusively allocated to one user or application, that is, resources are often shared among users and applications. Consequently, changes in the distributed system such as variation in demand of processor power, variation in number of available resources, or dynamic changes in the run-time behaviour of the application, hamper the efficient use of the metacomputing environment.

Consider, for example, an application that after a straightforward domain decomposition, can be mapped onto the processors of a parallel architecture. If the hardware system is homogeneous and allocated to only one application program, then the execution will run balanced until completion: we have mapped a static resource problem to a static resource system. However, if the underlying hardware system is a cluster of multi-user workstations we run into problems because the available processing capacity per node may change: in this case the static resource problem is mapped to a system with dynamic resources, resulting in a potentially unbalanced execution. Things can get even more complicated if we consider the execution of an application with a dynamic run-time behaviour on a metacomputer environment, i.e., the mapping of a dynamic resource problem onto a dynamic resource machine. The notion of redundant decomposition has been posed by de Ronde *et al.* [5] to introduce sufficient *richness* in parallel tasks to make a balanced workload in such a dynamic resource machine possible.

One way to deal with this dynamic changing resource requirement would be dynamically rebalance a job and its (parallel) constituents by migration of processes from overloaded to under-loaded resources at run-time. If the dynamic load balancing occurs locally, the wide-area scheduler does not participate. However, the local resource manager might request the wide-area scheduler that a job be re-scheduled elsewhere. The next section describes the design and implementation of these functionalities that are needed with dynamic load balancing, i.e., process migration of running (parallel) jobs.

4 Process Migration in Message Passing Environments

Process migration support can be incorporated at two operation levels: *operating system level* and *user level*. In operating system level implementations the resource management facilities are supported by the OS kernel. Examples of such systems are Mach, Sprite, and MOSIX [1]. User level designs and implementations of adaptive systems include dynamic resource management facilities by providing their own dynamic load balancing run-time support. Examples of user level designs are Condor [9] for sequential, and MPVM [3] for parallel application systems.

In the Polder project we have the following design constraints for the process migration facility:

- since we assume that the major computational resource is a scalable cluster environment, the application programming model must be based on message passing;
- it is essential we support a generic operating system, therefore the machine platform operating system should be Unix;
- by hiding the complexity in libraries, the dynamic load balance run-time support system must be incorporated at user level.

These design constraints has motivated the development of DynamicPVM [10] and DynamicMPI. Both message passing environments are generally available on many different platforms and allows for the extension of process migration into their libraries. Although the design considerations are equal for DynamicPVM and DynamicMPI, there are some differences in the implementation. PVM (as basis for DynamicPVM) is a message passing environment that also includes process creation and termination and other resource management functionalities such as primitives for the allocation and deallocation of resources. The MPI 1.1 definition however, does not include any hooks to resource management functionalities required with process migration. This has to be included in the MPI run-time support system, but must be transparently to the application programmer.

In the following discussion we briefly outline aspects of the DynamicPVM system and its use in the Polder metacomputer. A more complete description of the design and implementation of DynamicPVM is presented in [10].

4.1 Design Aspects of Process Migration in DynamicPVM

Process *migration* (operating system level and user level) is realized by the movement of an active process from one machine to another in a parallel or distributed computing system. The process is suspended and detached from its environment, its state and data (the *checkpoint*) transfered to the destination host, where it is restarted and attached to the destination environment. The major requirement for providing a migration facility is *transparency*: the execution of a process should proceed as if the migration never took place. In parallel application systems like PVM applications, this transparency should hold also for the migrated process's communication partners. The application programs then do not to have take account for possible complications of checkpointing and migration.

From the requirements defined above, it follows that DynamicPVM must incorporate a checkpoint/migration facility and location independent task identifiers, in order to support transparent process migration. The checkpoint/migration functionality in DynamicPVM is based on the ideas of the facility provided by the Condor system. DynamicPVM extended the checkpoint protocol to safely checkpoint communicating parallel tasks without loss of messages. The location independent task identifiers, or virtual task identifiers, guarantees a unique name space for tasks independent of their location. Thus the same task can be addressed with the same task identifier after migration. Compare the virtual task identifiers with virtual memory addresses: the virtual memory address can be mapped to different physical addresses during the execution of a program.

4.2 Implementation Aspects of DynamicPVM

The Scheduler. Although the scheduler is not considered as an integral part of DynamicPVM, its role and interface is mentioned here. In line with the top-down perspective of the wide-area and local site scheduler, resides the DynamicPVM scheduler beneath the local site scheduler. The local site resource manager is the authority that allocates the resources for the DynamicPVM cluster. The DynamicPVM scheduler acts as a resource manager within this DynamicPVM cluster, that is, it decides when to migrate a task and to which host it is moved. However, the DynamicPVM scheduler can request or relinquish resources in interaction with the the local resource manager. For example, the

Condor system defined an application interface, which is called CARMI [12], to support this interaction with the local resources manager.

In this scenario, the DynamicPVM scheduler largely determines the efficacy of the DynamicPVM system in its aim for load balancing. The development of good algorithms or heuristics for load balancing is a study on itself and is beyond the scope of this article. The current scheduler decides on (re-)allocation of processors for tasks, based on gathered load information of the workstation pool.

The scheduler is implemented as a normal PVM task. A consequence of implementing the scheduler as a PVM task, is that an additional interface must be provided to enable the scheduler to interact with the DynamicPVM system. To this end, the PVM library is extended with an interface routine, `pvm_move(tid, host)`, that initiates the migration of task `tid` to the specified `host`.

Consistent Checkpointing Through Critical Sections. To implement dynamic load balancing by task migration, the run-time support system must be able to create an image of the running process, the so-called *checkpoint*. A checkpoint of an active process consists of the state and data of the process, together with some additional information to recreate the process.

A complication with checkpointing communicating PVM tasks, is that the state of the process also includes the communication status of the socket connections. Thus, to save the state of the process, the interprocess communication must also be in a well-defined state. Since suspension of the related communicating task is not desirable, the task should not be communicating with another task at the moment a checkpoint is created. To prohibit the creation of process checkpoints during communication, we apply the notion of *critical sections* and embed all interprocess communication operations in such sections. Checkpointing can only take place outside a critical section. When a checkpoint signal arrives during the execution of a critical section, the checkpointing is deferred. We have implemented the checkpoint facility with two different strategies for storing the process's state and data: *direct* checkpointing via TCP/IP and *indirect* checkpointing via NFS. The flexibility in checkpoint strategies allows for experiments with different checkpoint/migration protocols in the metacomputer.

Virtual Task Identifiers. In message passing environments, the process identifier or task identifier, *task id* for short, is a unique identifier which serves as the task's address and therefore may be distributed to other PVM tasks for communication purposes. For this reason the *task id* must remain unchanged during the lifetime of a task, even when the task is migrated.

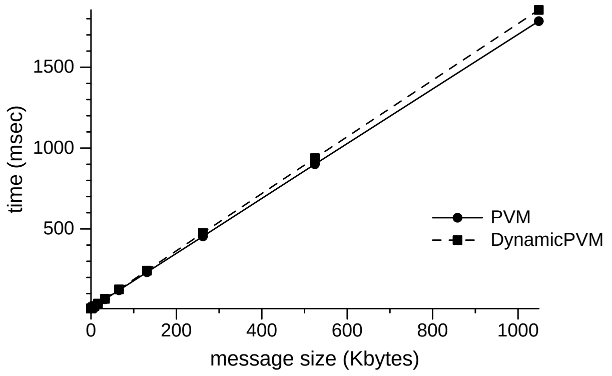
To provide transparent and correct message routing with migrating tasks, the *task ids* must be made location independent, thus by virtualizing the *task ids*. An important design constraint is that the routing facility must be highly efficient and should not impose additional limitations on the scalability. This is accomplished by maintaining additional routing information tables contained by all *pvmds*. These routing tables are consulted for all inter-task communication. Upon migration of a task, first the routing table of the master *pvmd* is updated to reflect the change in location of the migrated task. Next, the master *pvmd* broadcasts the routing table change to all other *pvmds*, such that each routing table reflects the actual location of all migrated tasks in the system.

5 Experiments with Dynamic Load Balancing in the Metacomputing Environment

5.1 Measuring DynamicPVM Overhead

By the design of DynamicPVM, two overhead factors are introduced: virtual task identifiers and task migration. Virtual task identifiers influence the communication performance, as the indirect addressing of tasks requires some extra administration. The task migration overhead is of importance for the scheduler, as the decision to migrate a task can depend on the (expected) migration time.

A well-known method to measure the basic communication properties of a message-passing systems is the *ping-pong* experiment. The measured time covers the transmission and the receipt of the message in user space. In this sense, the ping-pong experiment is a suitable benchmark to determine the overhead introduced by the DynamicPVM implementation. The ping-pong experiment was performed for both the public domain PVM implementation as well as the DynamicPVM implementation. The experiments were executed on a lightly loaded system of SPARCstation 4 workstations connected by a 10Mb/s Ethernet. The results of the ping-pong experiments are shown in Fig. 3. The gathered results are the mean of 30 independent experiments, in which for each message size 1000 messages are exchanged. The variance in the measurements is smaller than 1% of the mean.



Size (bytes)	PVM (msec)	DPVM (msec)
4	7.2	7.6
16	7.2	7.6
64	7.2	7.7
256	7.7	8.1
1 K	8.5	9.0
4 K	14.9	15.9
16 K	37.0	38.4
64 K	120.8	126.2
256 K	453.8	475.3
1 M	1785.1	1854.3

Fig. 3. PVM and DynamicPVM ping-pong results for low network load. The larger slope of the DynamicPVM curve shows a linear increasing overhead.

Figure 3 shows that the message send time in DynamicPVM increase faster than in PVM (the induced overhead is ±5% of the send time). The dependency of overhead on the message size is due to the routing table lookup for each packet sent between two *pvmds* in DynamicPVM. As the number of bytes increases, the number of packets sent also increases (see Section 4.2).

Figure 4 shows some results obtained by migrating a 75 Kbyte process with data segments of various sizes in both TCP and NFS checkpointing mode. The time measurements include the checkpoint of the active process, the migration, and the restart of the process. The results in Fig. 4 are the mean of 30 experiments, where the variance of the NFS mean is less than 5% and the variance of the TCP/IP mean is less than 1%.

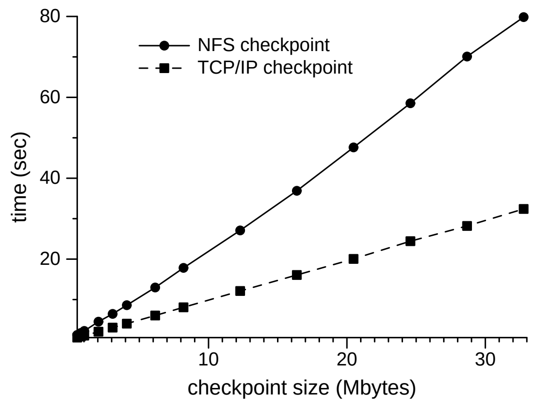


Fig. 4. Migration times for indirect NFS and direct TCP/IP checkpointing. The migration times increase linear with checkpoint size.

As can be seen in the figure, the time needed for the migration is linear to the size of the program. The tangent of the slope for NFS is ± 3, while the slope of TCP/IP is ± 1. This difference can be explained by the number of accesses to the checkpoint file in both strategies. The NFS checkpoint requires one write of the checkpoint to the file system, and a read and write for the creation of the new executable. With TCP/IP checkpointing, the new executable is created with a single state transfer over a socket connection. Nonetheless, both migration modes are efficiently implemented given the underlying protocol. For direct checkpointing, the measured throughput is almost 1 MB/s, while the bandwidth of Ethernet is 1 MB/s. With indirect checkpointing a throughput of about 400 KB/s is measured.

5.2 Adaptive Behaviour of DynamicPVM

DynamicPVM differs from PVM in its ability to adapt to changes in the workload of the available resources in cluster environments. The changes in workload can be induced by the irregular behaviour of the application or by activities of other user applications. Depending on the scheduler policy, the adaptive behaviour should be expressed in improved turn-around time of the parallel application, system utilization, etc. In this experiment we determine the turn-around time of PVM and DynamicPVM applications in a cluster environment with dynamically changing background workload.

To evaluate the qualities of DynamicPVM compared with PVM, the influence of the following four parameters should carefully be quantified:

– the dynamics of the temporal behaviour of the application (changes in workload);

– the spatial granularity of the application (problem size);
– the temporal granularity of the application (ratio communication/computation);
– the scheduler (different scheduling strategies).

The experiments must be designed to explore the parameter space such that trends can be detected and interrelations between the parameters appear.

A general problem with experiments in clusters of workstations is the lack of control of factors influencing the experiment. For consistent evaluation of the PVM and DynamicPVM systems, we need a controlled dynamic environment such that both systems endure the same amount of background workload during an experiment. The problem of the controlled dynamic environment is solved by creating a well-defined workload on a cluster of idle workstations.

Two independent clusters are configured for PVM and DynamicPVM, both clusters large enough to assign the tasks to different workstations. DynamicPVM needs also redundant workstations to balance workload—at any instant only one task is running per workstation. Although DynamicPVM uses a changing set of workstations to solve the application problem, per instant the number of workstations assigned to the problem is equal to PVM. During the experiment, each workstation endures the same amount of background workload, that is the total quantity is equal but the individual quantums are scheduled independently. The background jobs arrive with exponential interarrival times. The individual jobs take a fixed amount of work to complete, and are processor bound.

The PVM and DynamicPVM applications are executed in these controlled dynamic environments. During the execution in both environments, the PVM and DynamicPVM applications will endure the same *intensity* of background load.

In the pilot experiment we select two applications that reside at either end of the parallel application spectrum: EP and SSA. The EP benchmark falls in the category "embarrassingly parallel," requiring little communication between processors. The SSA (Systolic Simulated Annealing) application solves the crystalization of particles on a sphere, and puts a dynamic demand on the computational and the communication resources.

The EP experiments are pursued with two scheduling strategies. The workload is defined as the Unix system load average. Scheduling policy *sched 1* is based on workload difference in the DynamicPVM cluster: the workstation with the largest increase (decrease) in workload, is indicated as the most heavily (lightly) loaded host in the cluster. Based on these observations, the scheduler moves the tasks from the heavily loaded host to the lightly loaded host. Scheduling policy *sched 2* makes decisions based on the absolute workload. As the Unix system load average lags behind the actual workload distribution, the decision to migrate a task is delayed until the load average has draw level with the current situation—this gives a hurry after behaviour.

The influence of the background load on the EP benchmark resulted in an increase of the PVM execution time with 75% (increase from 0:52:34 to 1:33:48). The adaptive DynamicPVM system with scheduling policy *sched 1* brings the increased execution time back to 46%, with 58 task migrations. With scheduling policy *sched 2*, DynamicPVM is less successful to exploit the available resources. By its conservative behaviour (31 migrations), the tasks remain for a longer period on heavily loaded hosts before they are migrated. The DynamicPVM system succeeds in improving the performance of the EP benchmark, even with the 58 migrations in just over an hour.

Experiments with the SSA application are pursued with the conservative scheduler that uses a high threshold before a task is migrated (higher than with the EP benchmark). Several experiments with different problem sizes (number of particles on the sphere) were performed. The gain of DynamicPVM over PVM is around 10% for the different problem sizes, e.g., PVM runtime of 7:49:28 versus DynamicPVM runtime of 7:02:03 (with 4 migrations) for small problem instances, and PVM runtime of 17:31:46 versus DynamicPVM runtime of 16:12:44 (with 8 migrations) for large problem instances. These results show that large jobs with a conservative scheduler can take advantage of the DynamicPVM system with a limited number of migrations.

More experiments exploring the parameter space of the dynamics in workload, spatial and temporal granularity, and scheduling strategies will be showed in the presentation at the workshop.

6 Discussion and Conclusions

The presented research in adaptive systems and experiments with DynamicPVM shows that dynamic load balancing is a viable approach to the curse of dynamics in clusters of workstations. The communication and checkpoint overhead experiments in Section 5.1 indicate that the DynamicPVM system provides efficient task migration support. The experiments in a controlled highly dynamic cluster environment (Section 5.2) exposes the ability of DynamicPVM to react on changes in the cluster environment and reduce the turnaround time of the applications. The eventual success of DynamicPVM depends on the scheduling strategy.

The next step in our research is the integration of DynamicPVM into the Polder metacomputer. The key issue in the integration is the interplay of the DynamicPVM scheduler with the local resource manager. The design and implementation of more advanced scheduling strategies will be directed by experimental validation of the strategies in the resource management simulation model, which is developed within the Polder metacomputer framework.

The DAS distributed supercomputer provides an excellent experimental hardware platform to implement and validate different the designs of components in the Polder metacomputer. The DAS architecture with high speed local area and wide area network, grasps the characteristics of the prototypical metacomputer of the near future. In this respect, the various issues in wide-area scheduling and local site scheduling have to be included in the DAS resource management.

Open issues in the development of the Polder metacomputer are security and true *heterogeneous* distributed computing. Security determines the acceptance of metacomputing; if the resources of different organizational instances are gathered into a metacomputer, the integrity of the individual resources must be guaranteed. True heterogeneous distributed computing requires the support for process migration between different architectures. This heterogeneous process migration is difficult to solve at operating system level. At user level, we can take an object-oriented approach, and implement process/object migration into a library. The research in security and heterogeneous distributed computing aspects will be significant efforts in the future Polder metacomputer developments.

Acknowledgments

We express our thanks to Miron Livny (University of Wisconsin–Madison), Berry van Halderen (University of Amsterdam) and Judhi Santoso (University of Amsterdam/Institute of Technology Bandung) for the valuable discussions and the exchange of ideas with us. This research is supported in part by the Royal Netherlands Academy of Arts and Sciences under grant 95-BTM-15, and the Netherlands Organization for Scientific Research under grant 95MPR01/1.

References

1. A. Barak and O. La'adan. Performance of the MOSIX parallel system for a cluster of PC's. In *High Performance Computing and Networking (HPCN Europe '97)*, volume 1225 of *LNCS*, pages 624–635. Springer-Verlag, May 1997.
2. M. Brune, J. Gehring, and A. Reinefeld. A lightweight communication interface for parallel programming environments. In *High-Performance Computing and Networking (HPCN Europe '97)*, volume 1225 of *LNCS*, pages 503–513. Springer-Verlag, May 1997.
3. J. Casas, D. L. Clark, P. S. Galbiati, R. Konuru, S. W. Otto, R. M. Prouty, and J. Walpole. MIST: PVM with transparent migration and checkpointing. In *Third Annual PVM Users' Group Meeting*, Pittsburgh, PA, May 1995.
4. The distributed ASCI supercomputer (DAS). `http://www.asci.tudelft.nl/das/das.shtml`.
5. J. F. de Ronde, A. Schoneveld, P. M. A. Sloot, N. Floros, and J. Reeve. Load balancing by redundant decomposition and mapping. In *High Performance Computing and Networking (HPCN Europe '96)*, volume 1067 of *LNCS*, pages 555–561. Springer-Verlag, Apr. 1996.
6. T. A. DeFanti, I. Foster, M. E. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide area visual supercomputing. *International Journal of Supercomputer Applications and High Performance Computing*, 10(2/3):123–130, 1996.
7. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
8. A. S. Grimshaw and W. A. Wulf. Legion—A view from 50,000 feet. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 89–99, Syracuse, NY, Aug. 1996.
9. M. Litzkow, M. Livny, and M. W. Mutka. Condor—A hunter of idle workstations. In *8th IEEE International Conference on Distributed Computing Systems*, pages 104–111, 1988.
10. B. J. Overeinder, P. M. A. Sloot, R. N. Heederik, and L. O. Hertzberger. A dynamic load balancing system for parallel cluster computing. *Future Generation Computer Systems*, 12(1):101–115, May 1996.
11. The Polder metacomputing initiative. `http://www.wins.uva.nl/projects/polder/`.
12. J. Pruyne and M. Livny. Interfacing Condor and PVM to harness the cycles of workstation clusters. *Future Generation Computer Systems*, 12(1):67–85, May 1996.
13. M. S. Warren, M. P. Goda, D. J. Becker, J. K. Salmon, and T. Sterling. Parallel supercomputing with commodity components. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, Las Vegas, NV, June 1997.
14. J. B. Weissman and A. S. Grimshaw. A federated model for scheduling in wide-area systems. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pages 542–550, Syracuse, NY, Aug. 1996.