Network Communications in Grid Computing: At a Crossroads Between Parallel and Distributed Worlds

Alexandre Denis, Christian Pérez, Thierry Priol

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Network Communications in Grid Computing: At a Crossroads Between Parallel and Distributed Worlds

Alexandre Denis , Christian Pérez , Thierry Priol

N°4975

23rd October 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4975--FR+ENG

Network Communications in Grid Computing: At a Crossroads Between Parallel and Distributed Worlds

Alexandre Denis* , Christian Pérez† , Thierry Priol‡

Thème 1 — Réseaux et systèmes
Projet PARIS

Rapport de recherche n ° 4975 — 23rd October 2003 — 20 pages

Abstract: This paper studies a communication model that aims at extending the scope of computational grids by allowing the execution of parallel and/or distributed applications without imposing any programming constraints or the use of a particular communication layer. Such model leads to the design of a communication framework for grids which allows the use of the appropriate middleware for the application rather than the one dictated by the available resources. Such a framework is able to handle any communication middleware —even several at the same time— on any kind of networking technologies. Our proposed dual-abstraction (parallel and distributed) model is organized into three layers: arbitration, abstraction and personalities which are highlighted in the paper. The performance obtained with PadicoTM, our available open source implementation of the proposed framework, show that such functionality can be obtained with still providing very high performance.

Key-words: Computational grid, PadicoTM, middleware, communication framework

(Résumé : tsvp)

This work was supported by the Incentive Concerted Action "GRID" (ACI GRID) of the French Ministry of Research. Submited to: IEEE International Parallel and Distributed Processing Symposium (IPDPS 2004).

* Alexandre.Denis@irisa.fr
† Christian.Perez@irisa.fr
‡ Thierry.Priol@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Communications sur les grilles de calcul: au croisement des systèmes parallèles et répartis

Résumé : Cet article étudie un modèle de communication dont le but est d'étendre la portée des grilles de calcul en permettant l'exécution d'application parallèles et/ou réparties sans imposer de contraintes de programmation ou une couche de communication particulière. Un tel modèle nous amène à la conception d'une plate-forme de communication pour les grilles qui autorise l'usage de l'intergiciel adapté à l'application considérée plutôt que celui qui serait dicté par les ressources disponibles. Une telle plate-forme de communication supporte n'importe quel intergiciel communicant — éventuellement plusieurs en même temps — sur n'importe quel type de réseau. Le modèle que nous proposons, basé sur les deux abstractions parallèle et répartie, est organisé en trois couches : arbitrage, abstraction et personnalités, qui sont décrit dans cet article. La performance obtenue par PadicoTM, notre implementation de notre proposition de plate-forme de communication, montre qu'une telle fonctionnalité peut être obtenue tout en préservant de bonnes performances.

Mots-clé : Grille de calcul, PadicoTM, intergiciel, plate-forme de communication

1 Introduction

The emergence of computational grids as new high-performance computing infrastructures gives the users access to computing resources at an unprecedented scale in the history of computing. However, computational grids differ from previous computing infrastructure as they exhibit parallel and distributed aspects: a computational grid is a set of various and widely *distributed* computing resources, which are often *parallel* ranging from high-performance supercomputers to clusters of PCs. As a consequence, a grid usually contains various networking technologies— from SAN in a room through WAN at a continent scale.

As applications are *deployed* on grid resources, they ideally have to be able to adapt to their environment in general and to the networking environment in particular. The current programming practices associated with computational grids were strongly influenced by such adaptation capability. A common fashion is to see the grid as a virtual *parallel* computer so that programmers can follow the usual techniques of parallel programming, for example with MPI. Since MPI is available on a large number of networking technologies, applications based on this communication middleware will be able to adapt to the networking environment. However such adaptation is at the application programming interface level. Adaptation is also required at runtime. For example, an application linked with a MPI library configured to used the GM driver of a Myrinet network seriously constrains the application deployment only on those systems that provide such a network.

However, providing a single communication model (message-based), will not be enough for most applications as it does not take into account any external interactions such that visualization, steering, coupling of simulation codes or control interactions. Therefore, in addition to a parallel middleware system such as MPI, at least another middleware system is required to handle these new kind of interaction. Such a middleware system should be distributed oriented to handle dynamic connexion/deconnexion.

The first contribution of this paper is to propose a communication framework that decouple application middleware systems from the actual networking environment. Hence, applications become able to transparently and efficiently utilize any kind of communication middleware (either parallel or distributed-based) on any network that they are deployed on, removing thus the aforementioned deployment constraints. As a second contribution of this paper, the proposed model is able to concurrently support several communication middleware systems with very or no adaptation. Such capability is very important when using modern programming practices such as distributed component programming for the design of HPC applications. Indeed, dis-

tributed component models, such as CCA [2] or GridCCM [20], require to use both a communication middleware to let the components to communicate between each other and a communication runtime within a component if it encapsulates a parallel code. We have shown in [10], that even for standard networking technologies such as Ethernet with TCP/IP, having two communication middleware sharing such a network interface raises some serious technical concerns.

The remainder of this paper is divided as follows. Section 2 presents an analysis of grid communication based on some examples of typical grid usage. In section 3, we propose a communication framework model that supports both parallelism and distributed computing. Section 4 describes and evaluates the implementation of this model in the PadicoTM platform. Section 5 presents some related works. Finally, we conclude in Section 6.

2 Grid Communication Model Analysis

This section introduces some important features we think that actual and forthcoming grid-enabled applications should require. Then, it defines the communication paradigms and analyzes communication abstraction so as to draw the main directions for a communication framework for grids.

2.1 Grid Network Use Analysis

A grid application can be deployed on different resource configurations. For instance, one deployment may be a set of nodes within a single PC cluster equipped with a high-performance network, while another deployment may be a set of nodes in two separate PC clusters interconnected through a high-bandwidth WAN. Another example of grid use is given by parallel component based applications [2, 20] where a component embeds a parallel code. The component framework uses its own paradigm to interconnect components. This paradigm should be independent from the communication paradigms used internally by parallel components. Hence, a MPI-based component could be connected to a PVM-based component.

A last example is a grid application which support connexion and deconnexion from user to visualize and/or monitor the ongoing computation. Hence, the grid application is likely to used at least two middleware systems: one or more for managing its computation and another to manage the dynamic connexion/deconnexion of users.

These scenarios introduce some important features grid enabled middleware systems should support:

Transparency — The middleware systems used by an application should be able to transparently and efficiently use the available resources. For example, a MPI, PVM, Java or CORBA communication should be able to utilize high speed networks (SAN) as well as local area networks (LAN) and wide area networks (WAN). Moreover, they should adapt their security requirements to the characteristics of the underlying network, *eg.* if the network is secure, it is useless to cipher data.

Flexibility — There is a diversity of middleware systems, and we can assume there will always be. It seems important not to tie grid applications to a specific grid framework but instead to ease the "gridification" of middleware systems.

Interoperability — Grids are not a closed world. Grid applications will need to be accessible using standard protocols. So, there is a high need to keep protocol interoperability.

Support Multiple Communication Paradigms — Some programming models like parallel components (CCA [2], GridCCM [20]), or situations like a SOAP-based monitoring system of a MPI application, require several middleware systems. Thus, it is important to allow different middleware systems to be used simultaneously.

2.2 Communication Paradigm Analysis

If we define a communication *paradigm* as a family of middleware systems which are built on the same model, we can distinguish two important kinds of communication paradigms: the *parallel* paradigm and the *distributed* paradigm.

Parallel paradigm — The constraint is without no doubt high performance. Communications take place inside a definite and usually static set of nodes known by each other (mostly SPMD-oriented), messages have well-defined boundaries, the API is optimized for zero-copy implementations, there are collective operations which involves several nodes of the set. A typical example is MPI. We can distinguish distributed-memory parallelism and shared-memory parallelism; in this network-centric paper, we focus on distributed-memory parallelism.

Distributed paradigm — The constraint is interoperability. Connections are dynamic, managed on a per-link basis in a client/server way; interoperability is brought across architectures, operating systems and software vendors; communication primitives may use streaming. Some typical examples are TCP/IP, CORBA or SOAP.

These are *our definitions* and will be used in the remaining of this paper. They should be understood as a classification with soft boundaries, not as absolute rules; for example MPI 2 allows dynamic connections, a DSM system (Distributed Shared Memory) is not message-based, but we still consider them as parallel. In this paper, we consider TCP/IP and UDP, CORBA-IIOP [18], SOAP [4], HLA-RTI [15] and Java RMI as distributed-oriented; MPI, PVM, DSM, *FastMessage*, *Madeleine* [3] or *Panda* [22] as parallel-oriented.

2.3 Abstraction Level Analysis

The last step of our analysis deals with the different levels of communication abstraction found in a grid application.

We consider the abstraction of the resources as the definition of an *abstract* interface which is not bound to any particular implementation. There may exist several incarnations which implements the same abstract interface. Abstraction is a widely used mechanism to cope with the differences between various kinds of networks; in this case, it is called a *portability* mechanism. When an abstract interface is designed for portability and also to be used by several middleware systems and/or applications (and not only for the portability of one middleware system), it is a *genericity* mechanism. This results in a stack of software layers whose abstraction level increases down-top:

System-level — implemented by a network *driver* such as GM, BIP [21], VIA, *Sisci* [14] or other vendor-supplied communication library, or by the operating system such as TCP/IP sockets.

Generic-level — implemented by a *communication framework*, such as *Madeleine* [3], *Nexus* [12] or *Panda* [22]. The API, independent from the network, is likely to be used by a middleware system.

Application-level — implemented by a *middleware* system, such as CORBA, MPI, PVM or HLA-RTI. It implements a programming model. The API is designed to be used by applications.

3 A Model for Grid Communication Frameworks

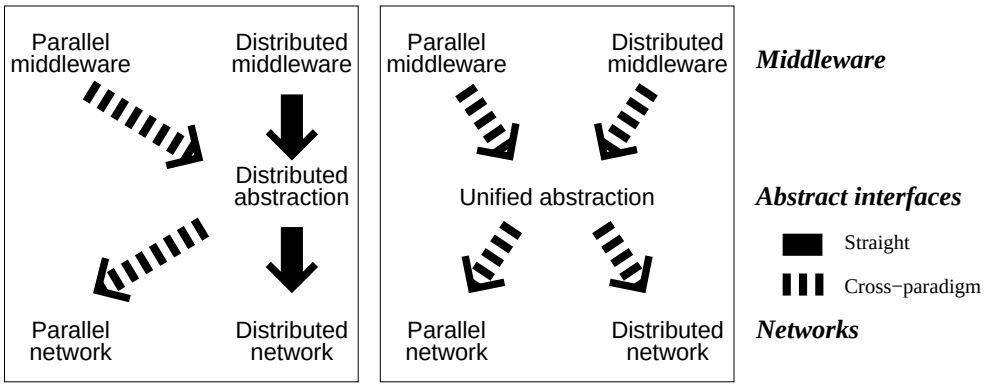
This section presents our proposed model of a communication framework for grids that takes into account both parallel and distributed paradigms.

3.1 Abstraction Model Study

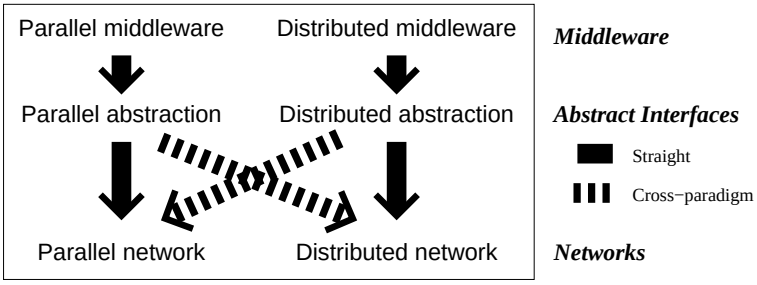
The commonly used abstraction model brings *portability*: the ability for a middleware system to utilize several kinds of networks, according to what is available. It brings also *genericity*: the ability to reuse the portability software infrastructure for several middleware systems. However, the genericity as it is usually achieved is based on the definition of a unique abstract interface. This choice is especially relevant for portability, but is questionable regarding genericity: this approach is generic *inside* a particular paradigm.

However, a lot of parallel middleware can utilize TCP/IP sockets that are a *distributed* abstract interface. This approach is well adapted for making a parallel system to look like a distributed network infrastructure (*eg.* Ethernet), but seems irrelevant to use a parallel-oriented network such as the internal network of a supercomputer or a cluster). As depicted in Figure 1 (a), the use of a single abstract interface imposes unnecessary compromises, even when running a parallel application on a parallel machine! For example, an MPI implementation built atop TCP/IP is able to run on most networking resources, including supercomputers networks, but is unable to utilize "parallel-specific" properties of these networks, such as optimized collective operations. This is due to the lack of expressiveness of the distributed-oriented TCP/IP API. Symmetrically, it is quite common to use a unique parallel interface on grids, for example MPICH-G2 [11]. It is possible to use it to implement distributed-oriented communication mechanisms, such as distributed objects. However, the parallel-oriented MPI interface cannot express properties which are essential for distributed computing, such as IP addressing, dynamic connections in a client/server fashion (not *spawn* as in MPI-2), or interoperability with other standard implementations. For instance, it seems impossible to build a standard-conforming CORBA implementation on top of MPICH-G2 (or more precisely, MPICH's abstract interface called "ADI-2") alone.

In both cases, a unique abstract interface biased towards only parallelism or distributed computing penalizes the middleware systems from the other paradigm since some properties available at system-level cannot be expressed by the abstract interface, so they are lost.



(a) Left side: everything expressed through a single abstraction (distributed) — two cross-paradigm translations for a parallel middleware atop a parallel network.
(b) Right side: a unified abstraction makes compromises in all cases! — gives up most possible optimizations and imposes compromises to everything.



(c) Dual-abstraction model: use different abstract interfaces for different paradigms — only required compromises are done.

Figure 1: Several abstraction models may be envisaged.

Thus we should try to find a better abstract interface which would combine both properties from parallelism and distributed computing, as depicted in Figure 1 (b); this abstraction would "keep the best of both worlds". In order to take into account the interoperability constraint from distributed computing, a unified abstract interface cannot be far from a distributed-oriented interface. More generally, it seems unrealistic to weaken the strong constraints of distributed computing to make them more look like the weaker hypothesis of parallelism which allow some optimizations: giving up the streaming capability from distributed computing in order to optimize a message-based communication system (*à la* MPI or *Madeleine*) breaks the required interoperability with TCP/IP; using topology and hardware configuration informations to optimize collective operations seems incompatible with the per-link connection management and interoperability with standard plain IP from the distributed side. A unified abstract interface cannot give up the strong constraints required by the distributed side, thus it uselessly imposes these strong constraints even to the parallel side. A single abstract interface, be it distributed, parallel, or unified, does not seems satisfactory.

Rather than trying to unify contrary things, we propose a dual-abstraction interface, with both a parallel- and distributed-oriented interface. Each middleware system is either parallel or distributed not both at the same time. For example CORBA, HLA and SOAP are distributed when MPI and PVM need a parallel abstract interface. There is no need to find an interface which would be both; it is sufficient to provide each middleware system with the appropriate abstract interface, and to supply each abstract interface on both kinds of networks. This dual-abstraction approach is depicted in Figure 1 (c). Each middleware system utilizes the required abstracted interface. Each abstract interface is instantiated on each network through an *adapter*: an adapter may be either *straight*, or *cross-paradigm*. Consequently, compromises for cross-paradigm translation are performed only when they are required. With such a dual-abstraction model, there always exists an abstract-level interface able to express the properties for each kind of hardware. Bending all system-level interfaces towards a unique abstraction does not seem appropriate because it loses some key features: a communication framework for grids cannot be parallel- nor distributed-only. We chose to build our grid communication framework on this dual-abstraction model.

3.2 Resource Virtualization for Seamless Swapping of Communication Methods

The middleware systems likely to be used by grid-enabled applications are various: MPI, CORBA, SOAP, HLA, JVM, PVM, etc. Moreover, for each kind of middleware,

there are several implementations which have their own specific properties. Developing a middleware system is a heavy task —for example, MPICH contains 200,000 lines of C— and requires very specific skills. Moreover, the standards —and thus, the middleware systems themselves— are ever-changing. It does not seem reasonable to re-develop an implementation of each one of these middleware systems specifically for a given communication framework. Instead, we chose to re-use existing implementations. Thus it is easy to follow the new versions and to use specific features of a given implementation.

To seamlessly re-use existing implementations of middleware systems, we choose to virtualize networking resources. It consists in giving the middleware system the illusion that it is using the usual resource it knows, even if the *real* underlying resource is completely different. For example, we show a "socket" API to a CORBA implementation so as to make it believe it is using TCP/IP, even if it is actually using another protocol/network behind the scene. This is performed through the use of thin wrappers on top of the appropriate abstract interface to make it look like the required API. We call these small wrappers *personalities*. It is possible to give several personalities to an abstract interface.

Virtualization and abstraction mechanisms with cross-paradigm adapters allows any middleware system to seamlessly utilize any network. However, even if a *straight* adapter is available, it is not always the better method, especially on distributed-oriented networks. The other methods are for example:

Parallel streams on WAN — Over a high-bandwidth high-latency WAN with TCP/IP, each single packet loss can dramatically lower the bandwidth. A solution consists in utilizing multiple sockets in parallel for a single logical link, so as to reduce the influence of each isolated loss. This principle of parallel streams is already used for example in *GridFTP* [1].

Online compression — On slow networks, it may be worth compressing data to speed-up the transfers. AdOC [16] implements an adaptive online compression mechanism.

Encryption and authentication — When a connection lays between two different sites, it is likely that the user wants authentication and/or encryption. This may be achieved through the use of a protocol plug-in. It raises a whole set of new problems, such as certificate management and credential delegation. We investigate the use of the Grid Security Infrastructure (GSI) [13] or IP*sec*.

Loss-tolerant protocol — On slow WAN which suffer from high loss-rate, applications may prefer to give up reliability against a better bandwidth, but not

accept totally uncontrollable losses. Such a tunable tradeoff is implemented in VRP [6], a protocol with a tunable loss tolerance.

These various communication methods may be supplied as *alternate adapters* beside *straight* and *cross-paradigm* adapters. They must exhibit the right abstract interface according to their respective paradigm. Their use is thus seamless from the point of view of the middleware systems. Thanks to these virtualization mechanisms, the hardware resources do not curb the programming model to be used in applications. The possible deployments schemes are more advanced than just parallel applications on a parallel machine or distributed applications on a distributed system. Each middleware system is able to use every available resources —parallel and distributed— with the most appropriate method — *eg.* CORBA as well as MPI are able to efficiently use Myrinet if available, or use WAN-specific methods if necessary. The virtualization enables the use of a communication paradigm not dictated by the hardware.

3.3 A Hybrid Parallel + Distributed Model

In this section, we propose a model of communication framework for grids, based on a 3-layer approach, with both parallel- and distributed-oriented abstract interfaces. An implementation of this model is depicted in Figure 2. Our proposed *dual-abstraction model* is organized in 3 layers: arbitration, abstraction, and personalities. Parallel and distributed paradigms are present at each level. Therefore, cross-paradigm translation is performed only when required (*ie.* distributed middleware atop parallel hardware or parallel middleware atop distributed networks) with no bottleneck of features.

Arbitration layer. Concurrent access to network hardware by multiple middleware systems at the same time is not straightforward. There is a high risk of access conflicts. We propose that arbitration should be dealt for at the lowest possible level, so as to build more advanced abstractions atop a fully reentrant system. Arbitration is performed by a layer which provides a consistent, reentrant and multiplexed access to every networking resources, each resource is utilized with the most appropriate driver and method. The arbitrated interfaces are designed for efficiency and reentrance. Thus, we propose these API to be callback-based (*à la* Active Message). For true arbitration, this layer is the only client of the system-level resources: all accesses to the network should be performed through the arbitration layer. It provides also arbitration between different networks (*eg.* Myrinet against Ethernet) so that they do not bother each other, and between different adapters (as defined in Section 3.1)

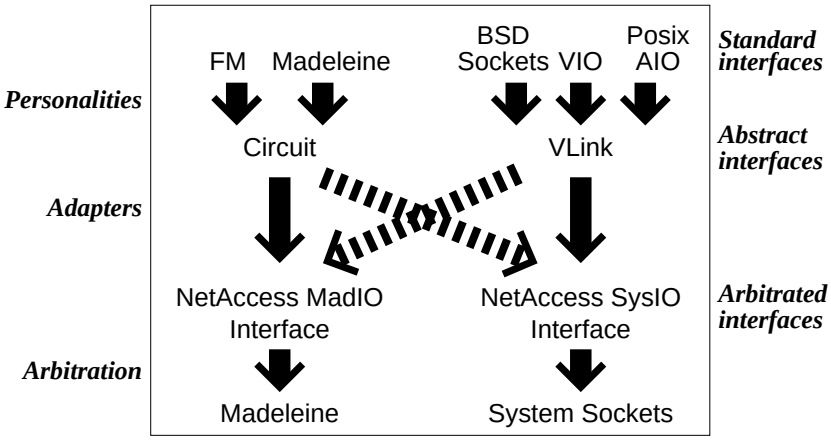


Figure 2: Implementation of the model in PadicoTM.

on the same network (*eg.* both CORBA and MPI on Myrinet) even if the communication library does not provide multiplexing. More details about cooperative access rather than competitive are given in [9].

Abstraction Layer. On top of the arbitration layer, we propose an abstraction layer which provides higher level services, independent from the hardware. Its goal is to provide abstract interfaces well suited for their use by various middleware systems. The abstract layer should be fully transparent: the interfaces are the same whatever the underlying network is. The abstraction layer supplies both parallel- and distributed-oriented abstract interfaces on top of every method from the arbitration layer, through modules called *adapters*. This layer is responsible for automatically and dynamically choosing the best available interface from the arbitration layer according to the available hardware; then it should map it onto the right abstract interface through the right *adapter*. As shown on Figure 2, adapters may be *straight* (same paradigm at system- and abstract-level, *eg.* parallel abstract interface on parallel hardware) or *cross-paradigm* — *eg.* distributed abstract interface on parallel hardware.

Personalities. In order to provide virtualized communication API, we propose a *personality* layer able to supply various standard APIs on top of the abstract interfaces. Personalities are thin wrappers which adapt a generic API to make it

look like another API. They do no protocol adaptation nor paradigm translation; they only adapt the syntax.

4 Implementation of the Communication Model

Padico [7] is our software infrastructure for Grid Computing. The communication model described in the previous section has been implemented in the high-performance runtime system of Padico called PadicoTM [9, 10] as depicted in Figure 2. The PadicoTM framework is used for parallel CORBA objects [8] and components [20]. This paper focuses only on the novel network model proposed in PadicoTM. However, PadicoTM addresses other issues for integrating different middleware systems, such as dynamic code loading and configuration, arbitration for multithreading, memory management and Unix signals. These other issues are purposely not discussed in this paper.

4.1 Network Access Arbitration: *NetAccess*

The arbitration layer in PadicoTM is called *NetAccess*, which contains two subsystems: *SysIO* for access to system I/O (sockets, files), and *MadIO* for multiplexed access to high-performance networks. A *core* handles a consistent interleaving among the concurrent polling loops. *NetAccess* is open enough so as to allow the integration of other subsystems beside *MadIO* and *SysIO* for other paradigms such as Shmem on SMP for example.

***NetAccess MadIO*: API for Accessing Parallel-oriented Hardware.** For good I/O reactivity and portability over high performance networks, we have chosen the high-performance network library *Madeleine* [3] as a foundation. *Madeleine* is used for high-performance networks such as Myrinet, SCI, VIA. *Madeleine* provides no more multiplexing channels than what is allowed by the hardware (*eg.* 2 over Myrinet, 1 over SCI). *MadIO* adds a logical multiplexing/demultiplexing facility which allows an arbitrary number of communication channels. Multiplexing on top of *Madeleine* adds a header to all messages. This can significantly increase the latency if not done properly. We implement *headers combining* to aggregate headers from several layers into a single packet. Thus, multiplexing on top of *Madeleine* adds virtually no overhead to middleware systems which send headers anyway. We actually measure that the overhead of *MadIO* over plain *Madeleine* is less than 0.1 μs which is imperceptible on most current networks.

***NetAccess SysIO*: API for Accessing Distributed-oriented Hardware.** Contrary to a widespread belief, using directly the socket API from the OS does not bring full reentrance, multiplexing and cooperation. Several middleware systems not designed to work together may get into troubles when used simultaneously, even with only plain TCP/IP. There are reentrance issues for signal-based I/O (used by middleware systems designed to deal with heavy load), which results in an incorrect behavior or worst in a crash. If a middleware system uses blocking I/O and another uses active polling, the one which does active polling holds near 100 % of the CPU time; it will result in inequity or even deadlock. To solve these conflicts, *SysIO* manages a unique receipt loop that scans the opened sockets and calls user-registered callback functions when a socket is ready. The callback-basedness guarantees that there is no reentrance issue nor signals to mangle with.

***NetAccess* core.** The *core* of *NetAccess* manages the threads with the polling loops. It enforces fairness between *SysIO* and *MadIO*. The interleaving policy between *SysIO* and *MadIO* is dynamically user-tunable through a configuration API to give more priority to system sockets or high performance network depending of the application.

4.2 Abstractions: *VLink* and *Circuit*

The abstract interfaces in PadicoTM are called *VLink* for distributed computing, and *Circuit* for parallelism.

Distributed abstract interface: *VLink*. The *VLink* interface is designed for distributed computing. It is client/server-oriented, supports dynamic connections, and streaming. In order to easily allow several personalities, *VLink* is based on a flexible asynchronous API. This API consists in five primitive operations —`read`, `write`, `connect`, `accept`, `close`. These functions are asynchronous: when they are invoked, they initiate (*post*) the operation and may return before completion. Their completion may be tested by polling the *VLink* descriptor; a handler may be set which will be called upon operation completion. Such a set of functions is called a *VLink*-driver. *VLink* drivers have been implemented on top of: *MadIO*, *SysIO*, Parallel Streams for WAN, AdOC [16], *loopback*.

Abstract interface for parallelism: *Circuit*. The *Circuit* interface is designed for parallelism. It manages communications on a definite set of nodes called a *group*.

A *group* may be an arbitrary set of nodes, *eg.* a cluster, a subset of a cluster, may span across multiple clusters or even multiple sites. *Circuit* allows communications from every node to very other node through an interface optimized for parallel runtimes: it uses incremental packing with explicit semantics to allow on-the-fly packet reordering, like in *Madeleine* [3]. Collective operations in *Circuit* still needs to be investigated. *Circuit* adapters have been implemented on top of *MadIO*, *SysIO*, *loopback* and *VLink* (to use the *alternates VLink* adapters); a given instance of *Circuit* can use different adapters for different links.

Selector. *VLink* and *Circuit* automatically choose which protocol to use according to a knowledge base of the network topology managed by PadicoTM and user-defined preferences. All protocols are available for both *VLink* and *Circuit* interfaces.

4.3 Personalities and Middleware Systems

PadicoTM provides several well-known API through simple "cosmetics" adapters over the *VLink* and *Circuit* abstract interfaces. These thin API wrappers are called *personalities*. The personalities for *VLink* are: *Vio* for an explicit use through a socket-like API; *SysWrap* supplies a 100 % socket-compliant API through wrapping at link stage for direct use within C, C++ or FORTRAN legacy codes without even recompiling. Thus, legacy applications are able to transparently use all PadicoTM communication methods without losing interoperability with PadicoTM-unaware applications on plain sockets. We implement an *Aio* personality on top of *VLink* which provides a plain Posix.2 Asynchronous I/O (*Aio*) API. Thin adapters on top of *Circuit* provides a *FastMessage* 2.0 API, and a (virtual) *Madeleine* API. Thanks to *SysWrap*, various middleware systems have been seamlessly ported on PadicoTM with no change in their code: CORBA implementations (omniORB 3, omniORB 4, ORBacus 4.0, all Mico 2.3.x including CCM-enabled versions), an HLA implementation (*Certi* from the *Onera*), and a SOAP implementation (gSOAP 2.2). A Java virtual machine (Kaffe 1.0.7) has been slightly modified for use within PadicoTM, with some changes in its multi-threading management code. Thanks to the *Madeleine* personality, the existing MPICH/Madeleine implementation can run in PadicoTM. The middleware systems are dynamically loadable into PadicoTM. Arbitration guarantees that any combination of them may be used at the same time.

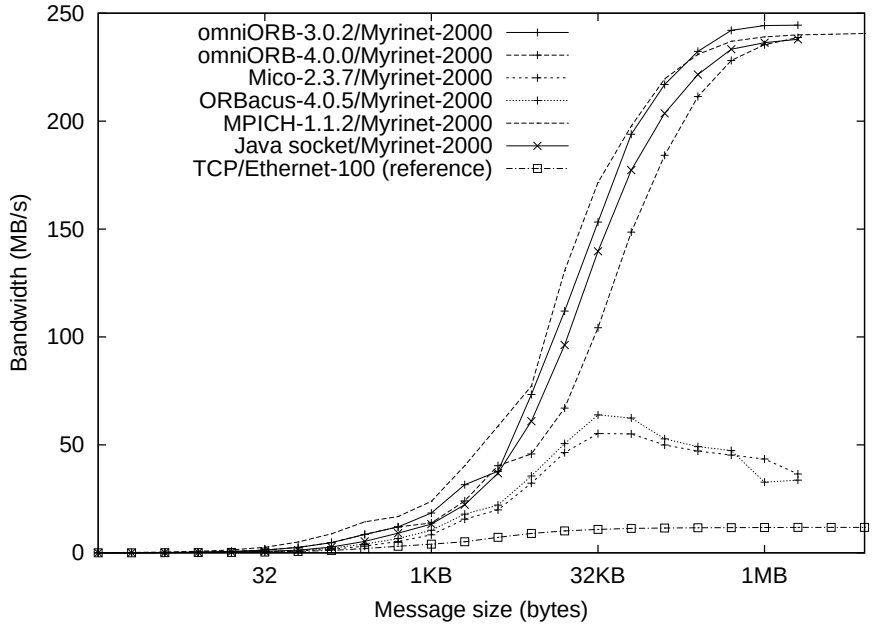


Figure 3: Bandwidth of various middleware systems in PadicoTM over Myrinet-2000.

4.4 Performance Evaluation

Our test platform is comprised of dual-Pentium III 1 GHz with 512 MB RAM, switched Ethernet-100, Myrinet-2000 and Linux 2.2. The raw bandwidth of various middleware systems in PadicoTM over Myrinet-2000 is shown in Figure 3. For MPI, omniORB and Java sockets, the peak bandwidth is excellent: roughly 240 MB/s, which is 96 % of the maximum Myrinet-2000 hardware bandwidth. The latency is 11 μs for MPI and 20 μs for omniORB. We notice the excellent performance for omniORB; as far as we know, omniORB in PadicoTM is the fastest existing CORBA implementation. Mico and ORBacus get lower performance because, unlike omniORB, they always copy data for marshalling and unmarshalling. Mico peaks at 55 MB/s with a latency of 63 μs, and ORBacus gets 63 MB/s with a latency of 54 μs. However, these poor performance results are due to bad internal design of the middleware systems themselves and are consistent with theory [9].

We have run test on VTHD, a French experimental high-bandwidth WAN. All middleware systems get roughly the same performance, namely a bandwidth of

9 MB/s and a 8 ms latency, which is the typical performance on this kind of network. When activating *Parallel Streams*, the bandwidth goes up to 12 MB/s which is the maximum possible given the fact that each node is connected to VTHD through Ethernet-100. On the WAN, every middleware systems get roughly the same performance since software overhead is negligible compared to the network speed.

We have tested VRP on a slow trans-continental Internet link. The link exhibits a typical loss-rate of 5-10 %. With TCP/IP and plain sockets, we get 150 KB/s; if we give up some reliability and allow up to 10 % loss with VRP, we get an average of 500 KB/s on the same link, *ie.* three times more.

5 Related Works

Several middleware environments for managing network communications have emerged. However, very few take both parallel and distributed paradigms into account and thus are not tailored for general grid applications. For example, *Panda* [22] is a framework designed for parallel runtimes, namely PVM and MPI. ADAPTIVE (ACE) [23] is a distributed-oriented generic communication environment *Harness* [17] and *Quarterware* [24] allow the use of multiple middleware systems at the same time; for the moment, they are limited respectively to MPI + PVM and MPI + RMI and published performance mention only plain TCP—no Myrinet nor WAN-optimized protocols. VMI [19] deals with both paradigms; its is close to VIA, and targets large clusters with SAN rather than WAN. Proteus [5] is a system for integrating multiple message protocols such as SOAP and JMS within one system. It aims at decoupling applications from protocols, which is an approach quite similar to ours, but at a much higher level in the protocol stack. Nexus [12] used to be the communication subsystem of Globus. Nowdays, it becomes accepted that MPICH-G2 [11] built on Globus-IO is a popular communication mechanism for grids, but only supports one API, namely MPI.

6 Conclusion

Grid applications can be largely leverage with in particularly an adequate support of middleware systems.

This paper has introduced a novel communication model for grids based on a crossroads of parallel and distributed worlds: both paradigms are present in the supported infrastructures and middleware systems. Hence, middleware systems are

decoupled from the actual network so that they can transparently and efficiently utilize any network they are deployed on.

The second advantage of the proposed communication model is its ability to support several middleware systems from different paradigms at the same time. This feature is very important for parallel objects/components programming models though traditional MPI application can also benefit from it.

The paper has also described the network-related aspects in the PadicoTM framework which implements this model and supports various methods to utilize the networks: BIP, GM, *Sisci*, VIA, TCP/IP, Parallel Streams, AdOC, VRP, and supports various middleware systems seamlessly: MPI, various CORBA implementations, HLA, SOAP, Java and a DSM.

Security issues need further investigations as they bring new problems. Other future works aim at bringing other communication methods for more flexibility to the deployment: tunnels for full-connectivity through firewalls, global addressing (without being tied to the IP system). PadicoTM is Open Source software and is available for download at `http://www.irisa.fr/paris/Padicotm/`.

References

[1] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. Data management and transfer in high performance computational grid environments. *Parallel Computing Journal*, 28(5):749–771, May 2002.

[2] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceeding of the 8th IEEE International Symposium on High Performance Distributed Computation*, Aug. 1999.

[3] O. Aumage, L. Bougé, A. Denis, J.-F. Méhaut, G. Mercier, R. Namyst, and L. Prylli. A portable and efficient communication library for high-performance cluster computing. In *IEEE Intl Conf. on Cluster Computing (CLUSTER 2000)*, pages 78–87, Technische Universität Chemnitz, Saxony, Germany, Nov. 2000.

[4] E. Cerami. *Web Services Essentials*, chapter Simple Object Access Protocol (SOAP), pages 49–112. O'Reilly & Associates, 1st edition, Feb. 2002.

[5] K. Chiu, M. Govindaraju, and D. Gannon. The proteus multiprotocol library. In *Proceedings of the 2002 Conference on Supercomputing (SC'02)*, Baltimore, USA, Nov. 2002.

[6] A. Denis. Variable Reliability Protocol: A protocol with a tunable loss tolerance for high performance over a WAN. Research Report RR2000-11, LIP, ENS Lyon, Lyon, France, Feb. 2000.

[7] A. Denis, C. Pérez, T. Priol, and A. Ribes. Padico: A component-based software infrastructure for grid computing. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[8] A. Denis, C. Pérez, and T. Priol. Portable parallel CORBA objects: an approach to combine parallel and distributed programming for grid computing. In *Proc. of the 7th Intl. Euro-Par'01 conf.*, pages 835–844, Manchester, UK, Aug. 2001. Springer.

[9] A. Denis, C. Pérez, and T. Priol. Towards high performance CORBA and MPI middlewares for grid computing. In G. A. Lee, editor, *Proc of the 2nd International Workshop on Grid Computing*, number 2242 in LNCS, pages 14–25, Denver, Colorado, USA, Nov. 2001. Springer-Verlag. In conjunction with *SuperComputing 2001 (SC'01)*.

[10] A. Denis, C. Pérez, and T. Priol. PadicoTM: An open integration framework for communication middleware and runtimes. In *IEEE International Symposium on Cluster Computing and the Grid (CCGRID2002)*, 2002.

[11] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, , and S. Tuecke. Wide-area implementation of the message passing interface. *Parallel Computing*, 24(12):1735–1749, 1998.

[12] I. Foster, J. Geisler, C. Kesselman, and S. Tuecke. Managing multiple communication methods in high-performance networked computing systems. *Journal of Parallel and Distributed Computing*, 40(1):35–48, 1997.

[13] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.

[14] IEEE. Standard for Scalable Coherent Interface (SCI). Standard no. 1596, Aug. 1993.

[15] IEEE. IEEE standard for modeling and simulation (M&S) high level architecture (HLA)—federate interface specification. IEEE Standard 1516, Sept. 2000.

[16] E. Jeannot, B. Knutsson, and M. Bjorkmann. Adaptive online data compression. Edinburgh, Scotland, July 2002.

[17] D. Kurzyniec, V. Sunderam, and M. Migliardi. On the viability of component frameworks for high performance distributed computing: A case study. In *IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edimburg, Scotland, July 2002.

[18] OMG. The Common Object Request Broker: Architecture and Specification V3.0. OMG Document formal/02-06-33, June 2002.

[19] S. Pakin and A. Pant. VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management. In *Workshop on Novel Uses of System Area Networks (SAN-1)*, Cambridge, Massachusetts, Feb. 2, 2002.

[20] C. Pérez, T. Priol, and A. Ribes. A parallel CORBA component model for numerical code coupling. In C. A. Lee, editor, *Proc. of the 3nd International Workshop on Grid Computing*, LNCS, Baltimore, Maryland, USA, Nov. 2002. Springer-Verlag. to appear.

[21] L. Prylli and B. Tourancheau. Bip: a new protocol designed for high performance networking on myrinet. In *1st Workshop on Personal Computer based Networks Of Workstations (PC-NOW '98)*, Lect. Notes in Comp. Science, pages 472–485. Springer-Verlag, apr 1998. In conjunction with *IPPS/SPDP 1998*.

[22] T. Rühl, H. Bal, R. Bhoedjang, K. Langendoen, and G. Benson. Experience with a portability layer for implementing parallel programming systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1477–1488, Sunnyvale, CA USA, AUG 1996.

[23] D. C. Schmidt. An architectural overview of the ACE framework: A case-study of successful cross-platform systems software reuse. *USENIX login magazine, Tools special issue*, Nov. 1998.

[24] A. Singhai. *Quarterware: a Middleware Toolkit of Software RISC Components*. PhD thesis, University of Illinois at Urbana-Champaign, 1999.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399