The MultiCluster Model to the Integrated Use of Multiple Workstation Clusters

Marcos Barreto*, Rafael Ávila**, and Philippe Navaux* **

Institute of Informatics — UFRGS
Av. Bento Gonçalves, 9500 Bl. IV
PO Box 15064 — 90501-910 Porto Alegre, Brazil
E-mail: {barreto,bohrer,navaux}@inf.ufrgs.br

Abstract. One of the new research tendencies within the well-established cluster computing area is the growing interest in the use of multiple workstation clusters as a single virtual parallel machine, in much the same way as individual workstations are nowadays connected to build a single parallel cluster. In this paper we present an analysis on several aspects concerning the integration of different workstation clusters, such as Myrinet and SCI, and propose our MultiCluster model as an alternative to achieve such integrated architecture.

1 Introduction

Cluster computing is nowadays a common practice to many research groups around the world that search for high performance to a great variety of parallel and distributed applications, like aerospacial and molecular simulations, Web servers, data mining, and so forth. To achieve high performance, many efforts have been devoted to the design and implementation of low overhead communication libraries, specially dedicated to fast communication networks used to interconnect nodes within a cluster, which is the case of Fast Ethernet [14], Myrinet [3] and SCI [12]. The design of such software is a widely explored area, resulting in proposals like BIP [21], GM [9], VIA [24] and Fast Messages [19].

Currently, there are other research areas being explored, such as administrative tools for cluster management and what is being called *Grid Computing*, with the objective of joining geographically distributed clusters to form a Metacomputer and taking benefit of the resulting overall computational power [4].

The work presented here is not focused on these areas directly, because our goal is to discuss a practical situation in which a Myrinet cluster must be interconnected with a SCI cluster to form a single parallel machine, which can be used to verify the application's behaviour when it runs on a shared memory cluster or on a message passing cluster, efficiently distribute tasks from an application according to their communication needs, offer a complete environment destinated to teach parallel and distributed

* M.Sc. student at PPGC/UFRGS (CAPES fellow)
** M.Sc. (PPGC/UFRGS, 1999); RHAE/CNPq researcher at PPGC/UFRGS
*** Ph.D. (INPG, Grenoble — France, 1979); Professor at PPGC/UFRGS

programming, allowing the user to express, through the same API, message passing and shared memory interactions.

This paper is organised as follows: Section 2 exposes an analysis on the problems that arise from integrating multiple workstation clusters; in Section 3 we present the MultiCluster model and the DECK environment as our contribution towards this objective; Section 4 brings some comments on related research efforts and finally Section 5 presents our conclusions and current research activities.

2 Integrating Multiple Clusters

When computer networks were an emergent platform to parallel and distributed programming, many efforts were dispended to solve problems related to joining individual PCs in a single virtual parallel machine. From these efforts, communication libraries such as PVM [8] and MPI [17] arose to allow individual network nodes to be identified within the parallel environment.

The integration of multiple workstation clusters presents a similar problem. Individual clusters of workstations are nowadays fairly well managed by communication libraries and parallel execution environments. When we start to think on clusters of clusters, again we have the same problems regarding the connection of elements that run independently from each other and still meet the compromise of offering to the user an appropriate environment for parallel and distributed programming. What we mean by appropriate is to provide an intuitive programming interface and offer enough resources to meet the programmer's needs.

As the purpose of this paper is to identify these problems and propose possible solutions to them, we have divided our study in hardware and software analysis.

2.1 Hardware Aspects

There are no major problems in the hardware point of view to achieve such integration, since the networks considered (Myrinet and SCI) could co-exist within the same node and use different techniques to communicate. Figure 1 presents the most simple cluster interconnection that could be realised.

Each individual cluster could have any number of physical nodes connected through a switch (in the Myrinet case) or directly as a ring (in the SCI case). To allow the integration, each cluster must have a "gateway" node configured with two network interfaces (two Myrinet NIs or a Myrinet + SCI NIs), where the additional Myrinet NI is used to link clusters. For the moment we do not consider SCI a suitable technology as a linking media, since a message-passing paradigm seems more adequate for this purpose.

2.2 Software Aspects

Several points have been discussed by the community in order to identify problems and solutions related to the design and implementation of communication libraries for cluster-based applications, with a main objective: provide high bandwith at small latencies. Besides this, the development of cluster middleware tools to furnish high availability and single system image support is an ongoing task [4, 11].

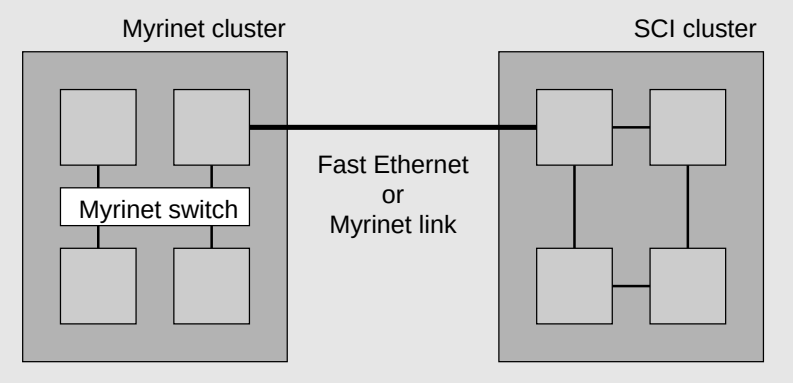

Fig. 1. The simplest way to interconnect two workstation clusters.

In the case of clusters of clusters, performance is not a key point due to the drawbacks implicitly imposed by the loosely coupled integration. There are other problems regarding such integration that must be attended first and performance will then be the consequence of the techniques used to solve them.

The first point to consider is how to combine message passing with distributed shared memory. A desirable solution would be to offer a single communication abstraction that could be efficiently implemented over message passing and shared memory architectures. In practice, however, it is easier to have an individual mechanism to each one and allow the user to choose between them, depending on his application needs.

Another point to treat is the routing problem, which arises when a task needs to exchange data with another task running in a remote cluster. It is necessary that the communication layer identifies what is the location of a communication endpoint and knows how to map physical nodes from separate clusters to be capable of routing messages between them.

Finally, heterogeneity could be a problem. Although most individual workstation clusters are internally homogeneous, there may be cases where multiple clusters could be heterogeneous in relation to each other. In these cases, problems regarding "endianisms" and floating-point data representation have to be addressed.

If the previous problems can be efficiently treated, it is also possible to provide the user with the capacity of deciding where to place a specific set of tasks, according to their communication needs. If the application granularity can be modelled considering the underlying platform, it is still possible to achieve good performance.

3 The MultiCluster Model

The MultiCluster model is an approach to join independent clusters and provide a simple programming interface which allows the user to configure and utilize such an integrated platform. With this model we intend to address and provide solution to the problems mentioned in the previous Section, while still keeping a well structured and

efficient programming environment. To best explain the proposed model, we have divided the discussion in hardware and software aspects.

3.1 Hardware Platform

We are assuming the configuration illustrated in Figure 1, which corresponds to our available hardware platform. We currently have a Myrinet cluster, composed by 4 Dual Pentium Pro 200 MHz nodes, and a SCI cluster, composed by 4 Pentium Celeron 300 MHz nodes. These clusters are linked through a Fast Ethernet network.

The choice of the media used to interconnect the clusters depends mostly on the application needs. It is possible to use a standard Ethernet link instead of Myrinet to realise the communication between clusters. We propose Myrinet as a link media because it could minimize the loss in performance originated by the integration of different platforms; for our model, however, it is enough that some node in each cluster plays the role of a gateway.

It is important to say that questions related to cost and scalability are out of the scope of this paper. In a near future, many companies and universities are likely to own a small number of cluster platforms, and so these questions are particular to each of them. We are assuming the situation where at least two clusters are available and have to be used together.

3.2 Software Structure

We have studied each problem mentioned in Section 2.2, trying to find the best solution to each one and structuring our software layer to carry out such solutions. As a result, the MultiCluster model follow some conceptual definitions which rule the way such integration must be handled.

Figure 2 shows the user-defined descriptor file to a MultiCluster application. In this file, the user must specify a list of machines within the clusters he wants to use, the communication subnets identifiers (used to inter-cluster communication), a set of logical nodes with its correspondents machines and the gateway nodes.

Physical and Logical Nodes. A physical node corresponds to each available machine plugged in any individual cluster and only matters to physical questions. Logical nodes are the set of available nodes from the application's point of view. In the case of message-passing clusters, each physical node corresponds to one logical node (this is mandatory). In shared-memory clusters, a logical node can be composed of more than one physical node. The distinction between logical nodes for Myrinet and SCI is made by the node id field. For example, "node 1:0" means the second node within the subnet 0 (which is Myrinet in our example), while "node 4:1" means the first node within the subnet 1 (which is SCI). It is important to notice that this numbering scheme, although complex, is entirely processed by the environment in a transparent manner; the user only knows how many logical nodes he has and what are the physical machines within each logical node.

```
// DECK user-defined descriptor file
// virtual machine
verissimo, quintana, euclides, dionelio,
scliar, ostermann, meyer, luft
// communication subnets
myrinet: 0
sci: 1
// logical nodes
node 0:0 machines: verissimo
node 1:0 machines: quintana
node 2:0 machines: euclides
node 3:0 machines: dionelio
node 4:1 machines: scliar, luft
node 5:1 machines: ostermann, meyer
// gateway nodes
gateways: quintana, scliar
```

Fig. 2. Descriptor file for a MultiCluster application.

Intra- and Inter-node Communication. As the application only sees logical nodes, it is relatively easy to adapt the different communication paradigms: inside a logical node, communication is made by shared memory; between logical nodes, communication is made by message passing. From the user's point of view, there is only one programming interface furnishing both mechanisms to specify communication over Myrinet or SCI clusters; the underlying communication layer is in charge of implementing one or another paradigm.

Heterogeneity. Although a less frequent problem, heterogeneity may arise depending on the availability of clusters that have to be interconnected. Here, we are considering different data representations and the need to indicate to the message receiver what is the architecture type of the message sender. This problem is implicitly treated by the communication software.

Even occuring some performance loss due to such integration, it is possible to the user to define the best location for his application tasks, creating communication resources according to each task location (i.e. communication subnets). Through this facility, the granularity of communication could be balanced among clusters, avoiding as long as possible the traffic across the link network.

3.3 The Programming Environment—DECK

The interface between the programmer and the MultiCluster architecture is the DECK environment. DECK (*Distributed Executive Communication Kernel*) is composed of a runtime system and a user API which provides a set of services and abstractions for the development of parallel and distributed applications. A DECK application runs in an SPMD style, split in terms of logical nodes.

DECK is divided in two layers, one called μDECK, which directly interacts with the underlying OS and a service layer, where more elaborate resources (including the support for multiple clusters) are made available. Figure 3 shows the layered structure of DECK.

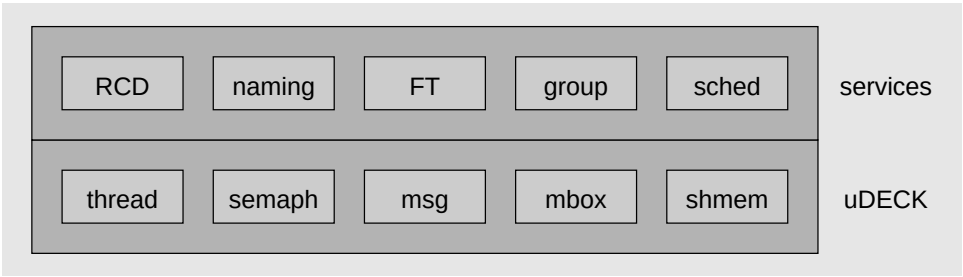

Fig. 3. Internal structure of DECK.

μDECK is the platform-dependent part of DECK. This layer implements the five basic abstractions provided within the environment: *threads, semaphores, messages, mailboxes* and *shared segments*. Each of these abstractions is treated by the application as an object, and has associated primitives for proper manipulation.

Messages present pack/unpack primitives, which do not necessarily perform marshalling/unmarshalling actions. When a message object is created, one of its attributes holds the identification of the host architecture. At the time of a pack no marshalling is performed; at the time of an unpack, if the receiving host is of a different architecture, the proper data conversion is made[1]. Messages can be posted to or retrieved from mailboxes. Only the creator of a mailbox is allowed to retrieve messages from it, but any other thread knowing the mailbox can post to it. To use a mailbox, the creator must register it in a naming server. There are two ways to obtain a mailbox address: fetching it in the name server or receiving it in a message.

The service layer is built on top of μDECK and aims to furnish additional, more sophisticated mechanisms that might be useful to the development of parallel applications, such as naming, group communication and fault tolerance support. In the scope of this paper, two elements of this layer must be analysed: the naming service and the Remote Communication Daemon (RCD).

The name server is a dedicated thread which runs in the first node within each cluster. For example, in the configuration illustrated in Figure 2, there will be a naming server running on “verissimo” and another running on “scliar”. Each naming server is responsible to register mailboxes created within its cluster. The name server is automatically executed when the application starts and has a well-known mailbox to allow other threads to communicate.

[1] It is important to observe that we only expect this to happen for messages crossing cluster boundaries, since clusters are assumed to be internally homogeneous.

The DECK/Myrinet Implementation. In the implementation of DECK on top of Myrinet, we are currently using BIP (*Basic Interface for Parallelism*) [21] as a communication protocol to efficiently use the underlying hardware and deliver high performance to applications. As BIP utilizes reception queues labeled with tags within each node, our mailbox implementation assigns a specific tag to each mailbox. To create a mailbox, the programmer uses a `deck_mbox_create()` primitive, passing as arguments the mailbox name and the communication subnet (defined in the descriptor file) in which this mailbox will be used.

The communication is made by post and retrieve operations, passing as arguments the corresponding mailbox and the message object, which contains the DECK supported datatypes. Posting a message is an asynchronous operation, while retrieving a message is a synchronous operation. To achieve this behaviour, we use the `bip_tisend()` and `bip_trecv()` primitives, respectively.

The implementation of μDECK mailboxes and messages on top of BIP is straightforward, since both are based on message passing. Shared segments, however, need an additional software DSM support to be implemented with the same library. For the moment we are studying the introduction of a DSM library, such as TreadMarks [25], to allow the usage of shared segments over Myrinet. The primitives for threads and semaphores are trivial and follow the Pthreads standard [13].

The DECK/SCI Implementation. We base our DECK/SCI implementation on two SCI programming libraries: Yasmin [23], which provides basic primitives for creation, mapping and synchronisation of shared segments, and Sthreads [22], which offers a Pthread-like environment on top of Yasmin.

A μDECK shared segment object offers primitives for creation, naming, mapping and locking. To the difference of Myrinet, SCI allows an easier implementation of both communication paradigms, so DECK/SCI offers mailboxes and messages as well as shared segments.

The creation of threads in DECK/SCI follows a simple round-robin placement strategy, according to the number of physical nodes that compose a logical node, which means that placement is still transparent to the end user. Notice that local memory can still be used for communication by local threads (i.e. threads in the same physical node), but it is up to the programmer to keep this kind of control. This means that, within SCI clusters, memory is only guaranteed to be correctly shared between remote threads if it is mapped into a μDECK shared segment.

RCD–Remote Communication Daemon. In order to support the MultiCluster model, the *Remote Communication Daemon* has been designed as a DECK service responsible for communicating to remote clusters. As each cluster must have a "gateway" node, the RCD is automatically executed inside this node when the application starts and follows the same semantic of the name server, i.e., it also has a well-known mailbox.

The RCD acts upon demand on two special cases: when fetching names defined remotely (i.e. on another cluster) and when posting messages to remote mailboxes. When a DECK primitive fails to fetch a mailbox address in a local name server, it contacts the RCD, which then broadcasts the request to other RCDs in the system and

wait for an answer, returning it to the caller. In the second case, when a DECK primitive sees a remote mailbox address when posting a message, it contacts the RCD, which then forwards the message to the RCD responsible for the communication subnet in which the mailbox is valid.

It is important to emphasize that communication between threads in different logical nodes, as well as different clusters, must always be made by message passing. Even in the case of a SCI cluster, there must be at least one mailbox to allow the communication with the RCD and, eventually, retrieve messages. For the moment we are disconsidering the utilisation of a global shared memory space to establish communication among clusters due to the lack of this support in the DECK/Myrinet implementation.

Our intention in designing DECK in three parts is to make it usable without changes in both single- and multi-clustered environments. In the first case, the RCD will simply not be brought into action by the application, since all the objects will be local to a specific cluster.

4 Related Work

Since the purpose of this paper is to discuss practical questions involved in the integration of multiple clusters and propose our model to achieve such integration, we tried to identify similar proposals regarding this subject.

There is a great number of research projects concerning the integration of multiple workstation clusters, such as NOW [1], Beowulf [2], Globus [7] and Legion [10]. The goal of these projects is to allow parallel and distributed programming over geographically distributed, heterogeneous clusters that corresponds to a “global computational grid”. The differential characteristic of our MultiCluster model is that we are assuming the simultaneous use of different network technologies, while these projects plans to use a common network technology to connect clusters, providing high scalability.

In terms of programming environments, there are also some efforts concentrated in joining message passing and distributed shared memory facilities, such as Stardust [5] and Active Messages II [16]. The main goal is to provide support for both message passing and distributed shared memory paradigms and, at same time, offer mechanisms to fault tolerance and load balancing support, as well as, portability. There are also some important contributions based on Java, such as JavaNOW [15], JavaParty [20] and Javelin [6]. All these contributions aims to provide distributed programming across networks of workstations or Web-based networks, differing in the communication model they used.

The idea behind MultiCluster is similar in some aspects with the objectives found in the projects/environments mentioned here, though in a smaller scale. Our research goal is to identify and propose solutions to problems related to specific integration of Myrinet and SCI clusters, while the goals of such projects comprise a larger universe, including fast communication protocols, cluster tools, job scheduling and so on.

Nevertheless, it is possible to state brief comparisons: our RCD is a simplest implementation when compared with Nexus, the communication system used inside Globus; it is just a way to give remote access to mailboxes defined in another clusters and allow us to separate the functionality of DECK when it runs in a single cluster platform.

The combination of message passing and distributed shared memory we offer is not so different than the usual mechanisms provided by the others environments. We want to efficiently implement these mechanisms in both clusters, without changing the programming interface. To accomplish this, our choice is to provide a mailbox object and a shared segment object to express message passing and memory sharing, respectively.

5 Conclusions and Current Work

In this paper we exposed some problems related to the integration of two different cluster platforms and proposed our MultiCluster model to achieve such desirable integration. We are developing our software environment aiming to accomplish a number of objectives, such as joining two specific cluster platforms (Myrinet and SCI) and providing a uniform API for parallel and distributed programming on both platforms, as well as opening research activities concerning such integration.

The integration is easier in terms of hardware because many solutions are already implemented within the OS kernel (e.g. co-existence of network device drivers). In terms of software, we have to decide what is the abstraction degree we want to offer to the programmer. It is important that the user be aware of the characteristics of each individual cluster to best adapt his application to take benefit of them. On the other hand, the DECK layer must abstract as much as possible implementation details, offering to the users a complete and simple API able to express the application needs. Currently, the descriptor file is the key point to configure the MultiCluster platform, because it represents the communication contexts and the logical nodes the user wants to use. Although this configuration is not so transparent, it is the most suitable way to adapt the execution environment according to the user needs. We consider that there are no problems in this task, since the execution environment guarantees the expected functionality.

Our work has been guided towards the design of a complete set of programming resources, enclosed in a software layer. Through the modularisation of DECK, we have divided our work in such way that we can parallelize our efforts to cover all problems exposed and to make available, as soon as possible, the MultiCluster model. At the moment we already have an implementation of DECK based on Pthreads and UNIX sockets, available at our Web page [18]. This implementation has played an important role to define the DECK structure and behaviour. At the time of this writing, we are concluding the implementation on top of BIP and collecting some performance results and, at same time, starting the implementation of DECK objects on top of SCI. The next step is to join both clusters and develop the RCD communication protocol.

References

1. T. Anderson, D. Culler, and D. Patterson. A case for NOW - Network of Workstations. Available by WWW at http://now.cs.berkeley.edu, Out. 1999.
2. Beowulf. The Beowulf project. Available by WWW at http://www.beowulf.org, Jun. 1999.
3. N. Boden et al. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

4. Rajkumar Buyya. *High Performance Cluster Computing*. Prentice Hall PTR, Upper Saddle River, NJ, 1999.
5. Gilbert Cabillic and Isabelle Puaut. Stardust: an environment for parallel programming on networks of heterogeneous workstations. *Journal of Parallel and Distributed Computing*, 40:65–80, 1997.
6. B. Christiansen et al. Javelin: Internet-based parallel computing using Java. Available by WWW at http://www.cs.ucsb.edu/research/javelin/, Nov. 1999.
7. Ian Foster and Carl Kesselman. The Globus project. Available by WWW at http://www.globus.org, Jul. 1999.
8. Al Geist et al. *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, MA, 1994.
9. GM message passing system. Available by WWW at http://www.myri.com, Nov. 1999.
10. A. Grimshaw et al. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), Jan. 1997.
11. Kai Hwang and Zhiwei Xu. *Scalable Parallel Computing: Technology, Architecture, Programming*. McGraw-Hill, New York, NY, 1997.
12. IEEE. IEEE standard for Scalable Coherent Interface (SCI). IEEE 1596-1992, 1992.
13. IEEE. Information technology—portable operating system interface (POSIX), threads extension [C language]. IEEE 1003.1c-1995, 1995.
14. IEEE. Local and metropolitan area networks-supplement—media access control (MAC) parameters, physical layer, medium attachment units and repeater for 100Mb/s operation, type 100BASE-T (clauses 21–30). IEEE 802.3u-1995, 1995.
15. Java and High Performance Computing Group. The JavaNOW project. Available by WWW at http://www.jhpc.org/projects.html, Nov. 1999.
16. Steven S. Lumetta, Alan M. Mainwaring, and David E. Culler. Multi-protocol Active Messages on a cluster of SMP's. In *Proc. of SuperComputing 97*, 1997.
17. MPI FORUM. Document for a standard message passing interface. *International Journal of Supercomputer Applications and High Performance Computing Technology*, 8(3/4), 1994.
18. The MultiCluster project. Available by WWW at http://www-gppd.inf.ufrgs.br/projects/mcluster, Nov. 1999.
19. S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages for Myrinet. In *SuperCOmputing '95*. IEEE Computer Society Press, 1996.
20. Michael Philippsen and Matthias Zenger. JavaParty: A distributed companion to Java. Available by WWW at http://wwwipd.ira.uka.de/JavaParty, Nov. 1999.
21. Loic Prylli and Bernard Tourancheau. BIP: A new protocol designed for high performance networking on Myrinet. In José Rolim, editor, *Parallel and Distributed Processing*, number 1388 in Lecture Notes in Computer Science, pages 472–485. Springer, 1998.
22. Enno Rehling. Sthreads: Multithreading for SCI clusters. In *Proc. of Eleventh Symposium on Computer Architecture and High Performance Computing*, Natal - RN, Brazil, 1999. Brazilian Computer Society.
23. H. Taskin. Synchronizationsoperationen für gemeinsamen Speicher in SCI-Clustern. Available by WWW at http://www.uni-paderborn.de/cs/ag-heiss/en/veroeffentlichungen.html, Aug. 1999.
24. VIA – Virtual Interface Architecture. Available by WWW at http://www.via.org, Nov. 1999.
25. Willy Zwaenepoel et al. TreadMarks distributed shared memory (DSM) system. Available by WWW at http://www.cs.rice.edu/~willy/TreadMarks/overview.html, Dez. 1998.