DEPLOYING APPLICATIONS IN MULTI-SAN SMP CLUSTERS

Albano Alves[1], António Pina[2], José Exposto[1] and José Rufino[1]

[1] *ESTiG, Instituto Politécnico de Bragança.*
{albano, exp, rufino}@ipb.pt

[2] *Departamento de Informática, Universidade do Minho.*
pina@di.uminho.pt

Abstract The effective exploitation of multi-SAN SMP clusters and the use of generic clusters to support complex information systems require new approaches. On the one hand, multi-SAN SMP clusters introduce another level of parallelism which is not addressed by conventional programming models that assume a homogeneous cluster. On the other hand, traditional parallel programming environments are mainly used to run scientific computations, using all available resources, and therefore applications made of multiple components, sharing cluster resources or being restricted to a particular cluster partition, are not supported.

We present an approach to integrate the representation of physical resources, the modelling of applications and the mapping of application into physical resources. The abstractions we propose allow to combine shared memory, message passing and global memory paradigms.

Keywords: Resource management, application modelling, logical-physical mapping

1. Introduction

Clusters of SMP (Symmetric Multi-Processor) workstations interconnected by a high-performance SAN (System Area Network) technology are becoming an effective alternative for running high-demand applications. The assumed homogeneity of these systems has allowed to develop efficient platforms. However, to expand computing power, new nodes may be added to an initial cluster and novel SAN technologies may be considered to interconnect these nodes, thus creating a heterogeneous system that we name multi-SAN SMP cluster.

Clusters have been used mainly to run scientific parallel programs. Nowadays, as long as novel programming models and runtime systems

are developed, we may consider using clusters to support complex information systems, integrating multiple cooperative applications.

Recently, the hierarchical nature of SMP clusters has motivated the investigation of appropriate programming models (see [8] and [2]). But to effectively exploit multi-SAN SMP clusters and support multiple cooperative applications new approaches are still needed.

2. Our Approach

Figure 1(a) presents a practical example of a multi-SAN SMP cluster mixing Myrinet and Gigabit. Multi-interface nodes are used to integrate sub-clusters (technological partitions).

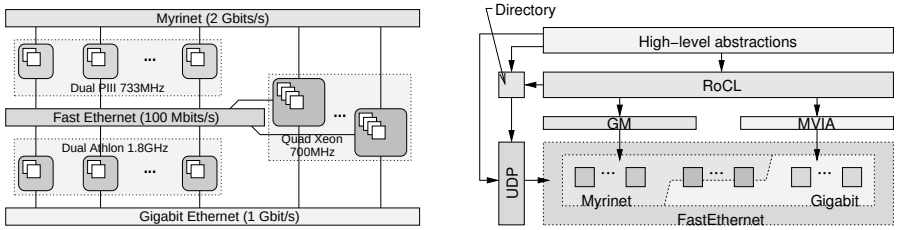

Figure 1. Exploitation of a multi-networked SMP cluster.

To exploit such a cluster we developed RoCL [1], a communication library that combines GM – the low-level communication library provided by Myricom – and MVIA – a Modular implementation of the Virtual Interface Architecture. Along with a basic cluster oriented directory service, relying on UDP broadcast, RoCL may be considered a communication-level SSI (Single System Image), since it provides full connectivity among application entities instantiated all over the cluster and also allows to register and discover entities (see fig. 1(b)).

Now we propose a new layer, built on top of RoCL, intended to assist programmers in setting-up cooperative applications and exploiting cluster resources. Our contribution may be summarized as a new methodology comprising three stages: *(i)* the representation of physical resources, *(ii)* the modelling of application components and *(iii)* the mapping of application components into physical resources. Basically, the programmer is able to choose (or assist the runtime in) the placement of application entities in order to exploit locality.

3. Representation of Resources

The manipulation of physical resources requires their adequate representation and organization. Following the intrinsic hierarchical nature

of multi-SAN SMP clusters, a tree is used to lay out physical resources. Figure 2 shows a resource hierarchy to represent the cluster of figure 1(a).

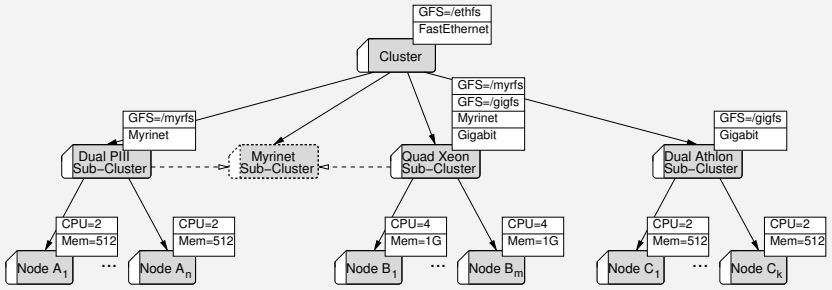

Figure 2. Cluster resources hierarchy.

3.1 Basic Organization

Each node of a resource tree confines a particular assortment of hardware, characterized by a list of properties, which we name as a domain. Higher-level domains introduce general resources, such as a common interconnection facility, while leaf domains embody the most specific hardware the runtime system can handle.

Properties are useful to evidence the presence of qualities – classifying properties – or to establish values that clarify or quantify facilities – specifying properties. For instance, in figure 2, the properties `Myrinet` and `Gigabit` divide cluster resources into two classes while the properties `GFS=...` and `CPU=...` establish different ways of accessing a global file system and quantify the resource *processor*, respectively.

Every node inherits properties from its ascendant, in addition to the properties directly attached to it. That way, it is possible to assign a particular property to all nodes of a subtree by attaching that property to the subtree root node. *Node* A_1 will thus collect the properties `GFS=/ethfs`, `FastEthernet`, `GFS=myrfs`, `Myrinet`, `CPU=2` and `Mem=512`.

By expressing the resources required by an application through a list of properties, the programmer instructs the runtime system to traverse the resource tree and discover a domain whose accumulated properties conform to the requirements. Respecting figure 2, the domain *Node* A_1 fulfils the requirements (`Myrinet`) $\wedge$ (`CPU=2`), since it inherits the property `Myrinet` from its ascendant.

If the resources required by an application are spread among the domains of a subtree, the discovery strategy returns the root of that subtree. To combine the properties of all nodes of a subtree at its root, we use a synthesization mechanism. Hence, *Quad Xeon Sub-Cluster* fulfils the requirements (`Myrinet`) $\wedge$ (`Gigabit`) $\wedge$ (`CPU=4*m`).

3.2 Virtual Views

The inheritance and the synthesization mechanisms are not adequate when all the required resources cannot be collected by a single domain. Still respecting figure 2, no domain fulfils the requirements (`Myrinet`) $\wedge$ (`CPU=2*n+4*m`)[1]. A new domain, symbolizing a different view, should therefore be created without compromising current views. Our approach introduces the original/alias relation and the sharing mechanism.

An alias is created by designating an ascendant and one or more originals. In figure 2, the domain *Myrinet Sub-cluster* (dashed shape) is an alias whose originals (connected by dashed arrows) are the domains *Dual PIII* and *Quad Xeon*. This alias will therefore inherit the properties of the domain *Cluster* and will also share the properties of its originals, that is, will collect the properties attached to its originals as well as the properties previously inherited or synthesized by those originals.

By combining original/alias and ascendant/descendant relations we are able to represent complex hardware platforms and to provide programmers the mechanisms to dynamically create virtual views according to application requirements. Other well known resource specification approaches, such as the RSD (Resource and Service Description) environment [4], do not provide such flexibility.

4. Application Modelling

The development of applications to run in a multi-SAN SMP cluster requires appropriate abstractions to model application components and to efficiently exploit the target hardware.

4.1 Entities for Application Design

The model we propose combines shared memory, global memory and message passing paradigms through the following six abstraction entities:

- domain - used to group or confine related entities, as for the representation of physical resources;
- operon - used to support the running context where tasks and memory blocks are instantiated;
- task - a thread that supports fine-grain message passing;
- mailbox - a repository to/from where messages may be sent/retrieved by tasks;
- memory block - a chunk of contiguous memory that supports remote accesses;

[1] n and m stand for the number of nodes of sub-clusters *Dual PIII* and *Quad Xeon*.

- memory block gather - used to chain multiple memory blocks.

Following the same approach that we used to represent and organize physical resources, application modelling comprises the definition of a hierarchy of nodes. Each node is one of the above entities to which we may attach properties that describe its specific characteristics. Aliases may also be created by the programmer or the runtime system to produce distinct views of the application entities. However, in contrast to the representation of physical resources, hierarchies that represent application components comprise multiple distinct entities that may not be organized arbitrarily; for example, tasks must have no descendants.

Programmers may also instruct the runtime system to discover a particular entity in the hierarchy of an application component. In fact, application entities may be seen as logical resources that are available to any application component.

4.2 A Modelling Example

Figure 3 shows a modelling example concerning a simplified version of SIRe[2], a scalable information retrieval environment. This example is just intended for explaining our approach; specific work on web information retrieval may be found eg in [3, 5].

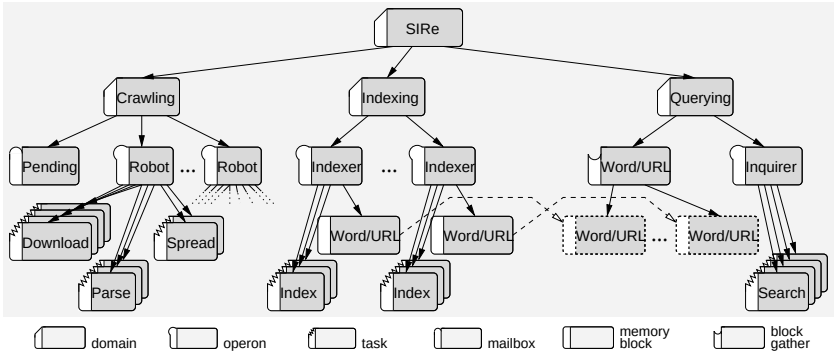

Figure 3. Modelling example of the SIRe system.

Each *Robot* operon represents a robot replica, executing on a single machine, which uses multiple concurrent tasks to perform each of the crawling stages. At each stage, the various tasks compete for work among them. Stages are synchronized through global data structures in the context of an operon. In short, each robot replica exploits an SMP workstation through the shared memory paradigm.

[2]A research supported by FCT/MCT, Portugal, contract POSI/CHS/41739/2001.

Within the domain *Crawling*, the various robots cooperate by partitioning URLs. After the parse stage, the spread stage will thus deliver to each *Robot* operon its URLs. Therefore *Download* tasks will concurrently fetch messages within each operon. Because no partitioning guarantees, by itself, a perfect balancing of the operons, *Download* tasks may send excedentary URLs to the mailbox *Pending*. This mailbox may be accessed by any idle *Download* task. That way, the cooperation among robots is achieved by message passing.

The indexing system represented by the domain *Indexing* is purposed to maintain a matrix connecting relevant words and URLs. The large amount of memory required to store such a matrix dictate the use of several distributed memory fragments. Therefore, multiple *Indexer* operons are created, each to hold a memory block. Each indexer manages a collection of URLs stored in consecutive matrix rows, in the local memory block, thus avoiding references to remote blocks.

Finally, the querying system uses the disperse memory blocks as a single large global address space to discover the URLs of a given word. Memory blocks are chained through the creation of aliases under a memory block gather which is responsible to redirect memory references and to provide a basic mutual exclusion access mechanism. Accessing the matrix through the gather *Word/URL* will then result in transparent remote reads throughout a matrix column. The querying system thus exploits multiple nodes through the global memory paradigm.

5. Mapping Logical into Physical Resources

The last step of our methodology consists on merging the two separate hierarchies produced on the previous stages to yield a single hierarchy.

5.1 Laying Out Logical Resources

Figure 4 presents a possibility of integrating the application depicted in figure 3 into the physical resources depicted in figure 2.

Operons, mailboxes and memory block gathers must be instantiated under original domains of the physical resources hierarchy. Tasks and memory blocks are created inside operons and so have no relevant role on hardware appropriation. In figure 4, the application domain *Crawling* is fundamental to establish the physical resources used by the crawling sub-system, since the operons *Robot* are automatically spread among cluster nodes placed under the originals of that alias domain.

To preserve the application hierarchy conceived by the programmer, the runtime system may create aliases for those entities instantiated

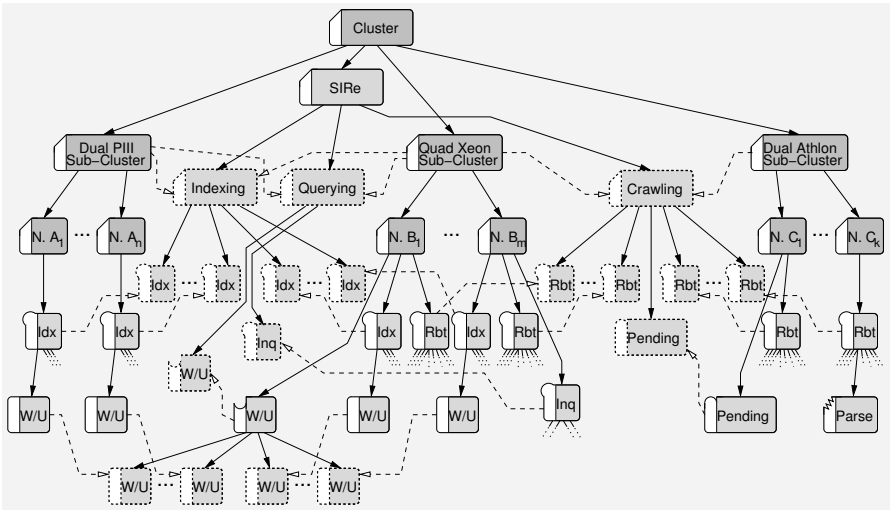

Figure 4. Mapping logical hierarchy into physical.

under original physical resource domains. Therefore, two distinct views are always present: the programmer's view and the system view.

The task *Parse* in figure 4, for instance, can be reached by two distinct paths: *Cluster* → *Dual Athlon* → *Node* C_k → *Robot* → *Parse* – the system view – and *Cluster* → *SIRe* → *Crawling* → $Robot^{(Alias)}$ → *Parse* – the programmer's view. No alias is created for the task *Parse* because the two views had already been integrated by the alias domain *Robot*; aliases allow to jump between views.

Programmer's skills are obviously fundamental to obtain an optimal fine-grain mapping. However, if the programmer instantiates application entities below the physical hierarchy root, the runtime system will guarantee that the application executes but efficiency may decay.

5.2 Dynamic Creation of Resources

Logical resources are created at application start-up, since the runtime system automatically creates an initial operon and a task, and when tasks execute primitives with that specific purpose. To create a logical resource it is necessary to specify the identifier of the desired ascendant and the identifiers of all originals in addition to the resource name and properties. To obtain the identifiers required to specify the ascendant and the originals, applications have to discover the target resources based on their known properties.

When applications request the creation of operons, mailboxes or memory block gathers, the runtime system is responsible for discovering a domain that represents a cluster node. In fact, programmers may specify

a higher-level domain confining multiple domains that represent cluster nodes. The runtime system will thus traverse the corresponding sub-tree in order to select an adequate domain.

After discovering the location for a specific logical resource, the runtime system instantiates that resource and registers it in the local directory server. The creation and registration of logical resources is completely distributed and asynchronous.

6. Discussion

Traditionally, the execution of high performance applications is supported by powerful SSIs that transparently manage cluster resources to guarantee high availability and to hide the low-level architecture eg [7]. Our approach is to rely on a basic communication-level SSI used to implement simple high-level abstractions that allow programmers to directly manage physical resources.

When compared to a multi-SAN SMP cluster, a metacomputing system is necessarily a much more complex system. Investigation of resource management architectures has already been done in the context of metacomputing eg [6]. However, by extending the resource concept to include both physical and logical resources and by integrating on a single abstraction layer *(i)* the representation of physical resources, *(ii)* the modelling of applications and *(iii)* the mapping of application components into physical resources, our approach is innovative.

References

[1] A. Alves, A. Pina, J. Exposto, and J. Rufino. RoCL: A Resource oriented Communication Library. In *Euro-Par 2003*, pages 969–979, 2003.

[2] S. B. Baden and S. J. Fink. A Programming Methodology for Dual-tier Multicomputers. *IEEE Transactions on Software Engineering*, 26(3):212–226, 2000.

[3] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[4] M. Brune, A. Reinefeld, and J. Varnholt. A Resource Description Environment for Distributed Computing Systems. In *International Symposium on High Performance Distributed Computing*, pages 279–286, 1999.

[5] J. Cho and H. Garcia-Molina. Parallel Crawlers. In *11th International World-Wide Web Conference*, 2002.

[6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *IPPS/SPDP'98*, pages 62–82, 1998.

[7] P. Gallard, C. Morin, and R. Lottiaux. Dynamic Resource Management in a Cluster for High-Availability. In *Euro-Par 2002*, pages 589–592. Springer, 2002.

[8] A. Gursoy and I. Cengiz. Mechanism for Programming SMP Clusters. In *PDPTA'99*, volume IV, pages 1723–1729, 1999.