Dynamic Resource Management in a Cluster for High-Availability

Pascal Gallard[1], Christine Morin[2], and Renaud Lottiaux[1]

[1] IRISA/INRIA – Paris Research group
[2] IRISA/Université de Rennes 1 – Paris Research group
{plgallar,cmorin,rlottiau}@irisa.fr

Abstract. In order to execute high performance applications on a cluster, it is highly desirable to provide distributed services that globally manage physical resources distributed over the cluster nodes. However, as a distributed service may use resources located on different nodes, it becomes sensitive to changes in the cluster configuration due to node addition, reboot or failure. In this paper, we propose a generic service performing dynamic resource management in a cluster in order to provide distributed services with high availability. This service has been implemented in the Gobelins cluster operating system. The dynamic resource management service we propose makes node addition and reboot nearly transparent to all distributed services of Gobelins and, as a consequence, fully transparent to applications. In the event of a node failure, applications using resources located on the failed node need to be restarted from a previously saved checkpoint but the availability of the cluster operating system is guaranteed, provided that its distributed services implement reconfiguration features.

1 Introduction

To efficiently execute high performance applications, cluster operating systems must offer some global resource management services such as a remote paging system[4], a system of cooperative file caches[7], a global scheduler[2] or a distributed shared memory[3, 1]. A cluster OS can be defined as a set of distributed services. Due to its distributed nature, the high-availability of such an operating system is not guaranteed when a node fails. Moreover, a node addition or shutdown should be done without stopping the cluster and its running applications.

In this paper, we propose a dynamic resource management service whose main goal is to hide any kind of change in the cluster configuration (node addition, eviction or failure) to the OS distributed services and to the applications, assuming process and page migration mechanisms are provided. A node failure should be also transparent for checkpointed applications. This work takes place in the framework of the design and implementation of Gobelins cluster OS. Gobelins is a single system image OS which aims at offering the vision of an SMP machine to programmers. Gobelins implements a set of distributed services for the global management of memory, processor and disk resources. Our generic

dynamic resource management service has been experimented with the global memory management service (a distributed shared memory)[6] of Gobelins for node addition and eviction.

In Section 2, we describe the proposed dynamic resource management service. Section 3 provides some details related to the service implementation and presents experimental results. Section 4 concludes.

2 Dynamic Resource Management

We call *configuration* the set of active nodes in the cluster. A node is considered to be *active* if it has not been detected failed by other active nodes and is not currently being added to or evicted from the cluster. A *configuration change* is due to a node addition, shutdown or failure. The cluster OS is said to be in the *stable state* when no configuration change is being processed. Otherwise, it is said to be in the *reconfiguration* state. Each of the distributed services that together form the cluster OS manages a collection of objects (for instance, a global memory management service manages a collection of memory pages). Objects may move between nodes at any time during the execution of an application on top of the cluster OS. A set of *metadata* is associated with each object. In particular the current location of an object is a metadata. In the model of distributed service we consider, each service implements a distributed directory with one entry per object, to store object metadata. On each node, the process responsible of the local directory entries is called a *manager*. In a given configuration, the manager of a particular directory entry is statically defined but when a configuration change occurs, the distribution of directory entries on the nodes belonging to the new configuration is updated.

The dynamic resource management service we have designed, called *adaptation layer*, is in charge of detecting configuration changes, updating the distribution of directory entries on cluster nodes in the event of a configuration change, triggering reconfiguration of distributed services when needed (for exemple after detection of a failure). Importantly, it is the adaptation layer which ensures that at any time, all cluster nodes have a consistent view of the current configuration. The adaptation layer is also used to locate directory managers[5]. At initialization time, each distributed service registers to the adaptation layer to benefit from its functions. The registration step allows distributed services to provide the adaptation layer service specific functions needed to perform the service reconfiguration. Note that the adaptation layer implements a single reconfiguration protocol to deal with any kind of configuration changes.

The adaptation layer is implemented by two processes on each node: the *locator* and the *supervisor*.

The locator process keeps track of directory managers for all distributed system services. It is activated each time an object is accessed by an application as the object metadata stored in the directory may be read to locate the considered object and/or updated depending on the operation performed on the object. The information used by the locator to locate managers is updated when the cluster

OS is in reconfiguration state. It does not change when the OS is in the stable state.

The supervisor process is responsible of the addition or the shutdown of the node on which it executes. It is the *supervisor* process that prepares its own node and notices the cluster. The set of *supervisors* in the cluster cooperate in order to maintain a consistent view of the cluster configuration. In this way, a node *supervisor* participates in the failure detection protocol. When a node failure happens (or is suspected) a consensus protocol, which is out of the scope of this paper, is executed. When a configuration change happens in the cluster, after a communication layer update, the *supervisor* triggers directory entries' migration. The functions registered by each service are used by the adapatation layer for the migration of directory entries.

3 Implementation in Gobelins and Evaluation

The dynamic resource management service described in the previous section has been implemented in Gobelins cluster OS and has been experimented with Gobelins global memory management service as an example of distributed service.

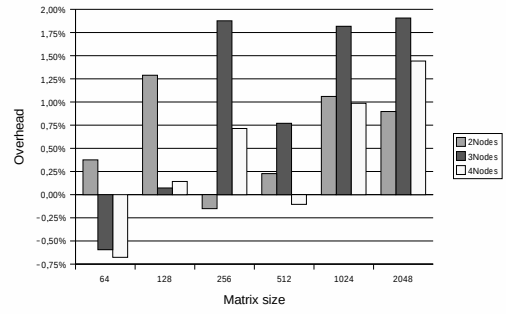

Fig. 1. Overhead evaluation

The cluster used for experimentation is made up of four *Pentium III* (500MHz, 512KB L2 cache) nodes with 512MB of memory. The nodes communicate with a Gigabit network. The Gobelins system used is an enhanced 2.2.13 Linux kernel. We consider here two of the Gobelins modules, the high performance communication system, and the global memory management system. We present in this paper an evaluation of the overhead due to the adaptation layer on the applications execution time. We have compared the execution time of the MGS application obtained with two different versions of Gobelins: the original one in which directory managers are located using a static modulo function (STAT) and a Gobelins version in which distributed services rely on the adaptation layer to locate directory managers (DYN).

The parallel application used in our tests is a Modified Gram-Schmidt (MGS) algorithm. The MGS algorithm produces from a set of vectors an orthonormal basis of the space generated by these vectors. The algorithm consists of an external loop running through columns producing a normalized vector and an inner loop performing for each normalized vector a scalar product with all the remaining ones. Time is measured on the external loop of the MGS program. Each

test is repeated 10 times. During the tests, error checking mechanisms in the communication layer were disabled.

We made several sets of experiments with different matrix sizes (64, 128, 256, 512, 1024 and 2048) on different clusters (2, 3 and 4 nodes). The figure presents the measured overhead for MGS calculated as: $overhead = \left(\frac{DYN}{STAT} - 1\right) * 100$.

In all cases, the overhead is less than 2%. In four cases (64-3N, 64-4N, 256-2N and 512-4N), the *dynamic version* is more efficient than the *static version*. As we indicate previously, the *static version* uses a distribution based on *modulo*. On another side, the *dynamic version* uses its own distribution that is different from *modulo*. In the particular case of Gram-Schmidt application, the new distribution decreases the number of page requests across the network. Cluster with two nodes and four nodes are similar cases because in these configurations every node has exactly the same number of directory entries to manage.

4 Conclusion

The proposed adaptation layer makes it possible to dynamically change the cluster configuration without stopping the OS services and consequently the running applications. In the future, we want to add some fault tolerance properties inside the directories in order to provide these properties to supported services.

References

1. C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared memory computing on networks of workstations. *IEEE Computer*, 29(2):18–28, 1996.
2. A. Barak and 0. La'adan. The MOSIX multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4-5):361–372, March 1998.
3. Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4), November 1989.
4. Michael J. Feeley, William E. Morgan, Frederic H. Pighin, Anna R. Karlin, Henry M. Levy, and Chandramohan A. Thekkath. Implementing global memory management in a workstation cluster. In *Proc. of the 15th ACM Symposium on Operating Systems Principles*, pages 129–140, December 1995.
5. Pascal Gallard, Christine Morin, and Renaud Lottiaux. Dynamic resource management in a cluster for scalability and high-availability. Research Report 4347, INRIA, January 2002.
6. R.Lottiaux and C.Morin. Containers : A sound basis for a true single system image. In *Proceeding of IEEE International Symposium on Cluster Computing and the Grid*, pages 66–73, May 2001.
7. Thomas E. Anderson, Michael D. Dhalin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. *ACM Transactions on Computer Systems*, 14(1):41–79, February 1996.