Load Sharing in Cluster Service Provision

R. Anane[1], R.J. Anthony[2], K-M. Chao[1] and M. Younas[1]
[1] *School of Mathematical and Information Sciences, Coventry University, UK*
[2] *Department of Computer Science, The University of Greenwich, UK*
r.anane@coventry.ac.uk

Abstract

Many of the factors that are driving the development and the adoption of grids, such as scale, dynamic and distributed characteristics and heterogeneity, are also pushing to the forefront service quality issues in grid computing. These include performance, reliability and security. Although grid middleware can address some of these issues, it has become imperative to ensure adequate service provision from compute servers.

This paper is concerned with the presentation of a load sharing scheme which can contribute to service provision in grid computing, at cluster level. As the aim of the scheme is to enhance performance and reliability, a proactive, non-preemptive and distributed approach was adopted. This ensures that load information is gathered continuously before it is needed, and that a task is allocated to the most appropriate node for execution. Reliability is enhanced by the decentralised nature of the scheme and the symmetric roles that the nodes can assume.

Keywords
Load sharing, proactivity, grid computing

1. Introduction

The level and intensity of the research devoted to grids and grid computing are a testimony to their increasing importance [1]. The development of middleware services to enable seamless access to remote and scarce resources has also been marked by concerns over the quality of service (QoS) issues. Reliability, security and performance figure prominently among these concerns. These relate to the distributed resources and services that makes up the grid as well to the service providers themselves, at local level. Within this context, the role of clusters as service providers in computational grids acquires special significance.

In this paper a load sharing scheme based on a non-preemptive and proactive approach is presented as a means of enhancing service quality at the local level. This performance enhancement strategy can be used to integrate the load sharing scheme with grid computing.

The paper is organized as follows. Section 2 gives an introduction to grids and grid computing. Section 3 presents some characteristics of load sharing. Section 4 draws a comparison between grid computing and load sharing. Section 5 introduces the load sharing scheme and describes its architecture. Section 6 presents the main features of the proactive approach. Section 7 concludes the paper and points to further work.

2. Grid computing

There is a general consensus that grids are large, dynamic, distributed and heterogeneous resource systems, and that grid computing is aimed at providing transparent access to scarce resources [2]. Among the different types of grid that have been proposed and identified, computational grids have received most attention. Computational grids are concerned with the efficient execution of tasks on a set of compute servers. In common with other types of grid the leveraging and the integration of available resources involve a set of fundamental middleware services: resource advertisement and discovery, scheduling and transfer of tasks [3].

In advertising its resources a compute resource provider might be required to include the number of nodes in a cluster, the amount of memory, operating system characteristics and probably load information. From another perspective, resource discovery is often performed by a directory service or a matchmaker. This service is designed to locate remote services and to help route computational requests to the most suitable compute resource in a grid [4]. Suitability can be expressed in terms of static information such as architecture and performance, or in terms of dynamic information such as availability and load. A global scheduler can provide this service. The transfer of tasks requires a suitable protocol and may involve negotiation [5] to secure commitment to service provision.

The emergence of grid computing has also been accompanied by the requirement for frameworks that can provide adequate quality of service. Related quality of service (QoS) issues include reliability, availability, security and computational response time [6].

3. Load sharing characteristics

Load sharing schemes in clusters are aimed primarily at detecting idle nodes and allocating tasks to them for execution. The development of load sharing schemes in distributed systems is motivated by three main factors [7]:

1. A significant differential load may exist between processing nodes.
2. The unused computing power in a network can be exploited by migrating tasks from overloaded nodes to underloaded or to idle nodes.
3. Significant performance gains can be achieved by migrating tasks.

Four main functions are usually required in load sharing schemes: information generation, selection of tasks, selection of location and transfer of tasks**.** Each of these functions is governed by a specific policy. The information policy is concerned with generating load information, determining the load index and disseminating this information. The selection policy is concerned with scheduling and involves the selection of tasks for transfer. The location policy is responsible for determining where to send a task, whereas the transfer policy determines when and how a transfer should take place. The second function of this policy is often redundant, being implicitly defined by the transfer mechanism.

A load sharing scheme can follow a reactive or proactive strategy. In a reactive load sharing strategy, the selection, location and transfer policies are activated by load imbalance or by a shortage of resources typically determined by comparing load metrics against threshold values. A tight coupling exists between the selection policy and the information policy. Reactivity is also characterised by migration decisions which are often based on a narrow information base such as the CPU queue length [8, 9, 10].

A proactive load sharing strategy seeks to prevent load imbalance by building a wide information base for making scheduling decisions and thus avoid wasteful transfers. It ensures that tasks are placed at the most appropriate node for execution. The load measure, in this case, usually represents a multidimensional analysis of resource availability. A proactive strategy is also marked by the decoupling of the information policy and the selection policy. The selection policy can therefore initiate scheduling decisions quickly, independently of the information policy.

Another dimension of load sharing schemes refers to the type of scheduling policy which can be either preemptive or non-preemptive. Where scheduling is preemptive the focus is on the selection, transfer and location policies. The scheduling is flexible; tasks can be transferred at any point in their execution. Preemptive scheduling, however, requires a complex mechanism for gathering the state of a process, transferring the state to the target node and re-instating the task. Because the state of a process is distributed throughout various structures of an operating system, the implementation of a preemptive transfer mechanism requires significant kernel modification. Sprite [11] is an example of a load sharing scheme that also follows a reactive approach.

Non-preemptive load sharing puts more emphasis on the information policy. Tasks can only be transferred at task initiation time. This promotes a proactive strategy. The selection policy only needs to determine whether a newly arrived task is suitable for transfer. The transfer policy determines whether or not a sufficient load differential exists between nodes to justify a transfer. This is often decided by comparing a simple load index against a threshold value. Non-preemptive load sharing requires a much simpler mechanism than its preemptive counterpart. A process is not created until the transfer is complete; there is no state to collect and transfer. Because of this simplification, non-preemptive load sharing can be implemented entirely at user level, as in Utopia [12].

4. Grid computing and load sharing

From the above presentation, it is obvious that grid computing and load sharing are both motivated by the requirement for access to remote resources. The two architectural frameworks are articulated along four fundamental services: resource advertisement or dissemination, resource discovery or identification, resource scheduling or selection and finally, transfer of tasks. Their corresponding underlying infrastructures are dynamic, and although they both involve some code mobility, they present some fundamental differences. A grid is large with a high degree of heterogeneity, and is more dynamic in its structure and availability, whereas a load sharing scheme is usually associated with a cluster with a relatively high degree of homogeneity. A lack of homogeneity may lead to stricter security constraints, and can be accompanied by a relatively low reliability. Furthermore, it may not support a preemptive scheduling policy. The potential for accessing resources is greater in grid computing, where they are rare and limited, than in load sharing schemes, where resources are similar but may be idle. In grid computing the emphasis is put on static and dynamic information, whereas in load sharing, a cluster has a relatively stable configuration, but with highly dynamic patterns of usage and behaviour. The scheduling function is more complex in grid computing than in load sharing. This difference is further accentuated by the time scale over which they operate and, the freshness of the information that is gathered or generated and then disseminated.

The dynamic nature of the grid and network latency may be an obstacle to the provision and the availability of accurate and up-to-date information. Information is expressed in terms of averages and trends. This feature may have an adverse effect on the availability of resources and on performance if load information, for example, is inaccurate. A load sharing scheme, on the other hand, relies on the generation of instantaneous information. Both systems can accommodate predictive models.

Despite their differences, however, the two approaches to distributed computing are to a large extent complementary. A load sharing scheme can be seen as a service within a compute resource, over which the grid middleware has no direct control. A load sharing scheme can, however, enhance the quality of service in grid computing by ensuring the efficient use of resources and the optimization of the performance of the compute servers. The scheme relies on quality information that is readily available, accurate, relevant and up-to-date.

In the following section a load sharing scheme is presented. Its function is to optimise the use of resources by allocating tasks to nodes in a cluster based on instantaneous information. As the aim of the scheme is to enhance performance and reliability, a proactive, non-preemptive and distributed approach was adopted.

5. A load sharing scheme

Concert is a load sharing scheme with the aim of improving the performance of distributed systems, measured in terms of the average response-time of tasks, by placing tasks at the most suitable node for execution. Concert implements a non-preemptive transfer policy within a proactive load sharing strategy, in order to reduce the occurrence of the symptoms of load imbalance.

5.1. Design requirements

One of the requirements of the design of the scheme is that proactivity, performance and reliability should be achieved within a sound design and implementation framework. The design should conform to the following criteria:

- Decentralisation. Nodes should operate autonomously on locally accessible information. A centralised design should be avoided as it can lead to the creation of bottlenecks and single points of failure.
- Efficiency. The design should minimise, on one hand, the communication between nodes and between the modules that implement the scheme, and on the other hand, the processing overheads associated with information generation and dissemination.
- Reliability. Processing nodes should not rely on services provided by other nodes, and should be sufficiently autonomous to permit failure independently of each other.
- Transparency. The system should be transparent to users and to tasks. Users should not be aware of the existence of the load sharing scheme, and tasks should not be modified to accommodate the operation of the load sharing scheme.

5.2. Information requirements

One aspect of the environment of a distributed system is that it exhibits some intrinsic characteristic behaviour that can be monitored, and that its capacity to support the role a load sharing scheme can be enhanced. Two fundamental assumptions form the basis of the proactive strategy in Concert:

1. Tasks are executed more than once on nodes; records of tasks' behaviour can be compiled and stored for future reference. Predictive models can be supported.
2. The environment is a rich source of information and is responsive; mechanisms can be set up to probe the environment effectively.

The load sharing scheme interacts with the environment meaningfully, at regular intervals, with the aim of generating the required information and ensuring that it conforms to the following criteria:

- Availability. Network-wide information must be available locally and immediately.
- Relevance. The information must be relevant to, and support the goals of, load sharing.
- Accuracy. Accurate information is a requirement for effective scheduling. In Concert it is achieved by accessing kernel-generated information via new system calls and by exchange of information between nodes.
- Freshness. The system should generate up-to-date information, incorporate mechanisms for dealing with obsolete information, and ensure that updates occur when required.

The basic environment is provided by the Linux operating system, which is probed by the information policy of the load sharing scheme. The network is transparent to the user and the main objects of interest are tasks and nodes.

5.3. Node-level scheme architecture

The distributed architecture of Concert was achieved by implementing the traditional four-policy structure of a load sharing scheme in each node. Although some of these policies may be amalgamated into a common module, conceptually they can still be distinguished. The information policy is implemented in the form of the Resource Availability Daemon (RAD) and the Resource Utilization Daemon (RUD). The selection, transfer and location policies are all incorporated in the Distributed Scheduler (DS). This combination increases efficiency by reducing communication between the different components. In this implementation, the transfer policy is largely redundant since only non preemptive transfers are supported, and transfers can only occur when a new task arrives. Transfers are initiated directly by the Distributed Scheduler. In addition to this fundamental structure, Concert includes another module, the Remote Task Manager (RTM), which manages the execution of migrated tasks. The Remote Task Manager acts as an abstraction of the remote kernel to client nodes from where tasks were transferred. The scheme architecture at node level is shown in Figure 1.

The implementation modules are now discussed. The Resource Availability Daemon has three functions:

1. measuring local load level, using a new system call to extract kernel-generated information;
2. disseminating load information to other nodes within the cluster; and
3. maintaining a database of load information concerning the local and remote nodes. This information is made available to the Distributed Scheduler on demand.

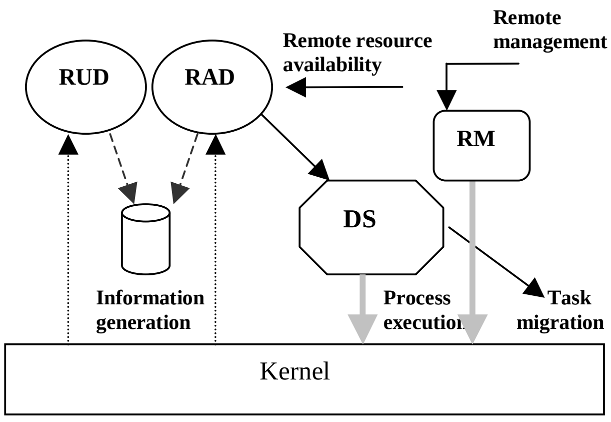

Figure 1. Node-level architecture

The Resource Utilisation Daemon records the resource usage of tasks. Kernel-generated information is extracted via a new system call and used to update a database of tasks' resource requirements information.

The Distributed Scheduler is a task scheduler operating at the cluster level. A task may be scheduled for execution at any processing node in the cluster. The Distributed Scheduler decides where to execute task (locally or at a specific remote site). This decision is based on locally held information concerning the load level at processing nodes (obtained from the Resource Availability Daemon database) and concerning the resource requirements of tasks (obtained from the Resource Utilisation Daemon database). The Distributed Scheduler directly dispatches the task for execution. When a task is to be executed remotely, the Distributed Scheduler sends a task-transfer message to the Remote Task Manager at the appropriate remote node.

5.4. Implementation

Concert is implemented in C on a Linux system. The implementation consists of four user-space modules, kernel modifications and bash shell modifications. Communication between the user-space modules at a node is achieved through remote procedure calls and access to shared data files. Communication between modules at different nodes is achieved through message passing, implemented using UDP. Communication between user-space modules and the operating system is achieved via new system calls. In each case the choice of communication mechanism is motivated by the need to create an efficient and robust system. The modules that implement the information policy, the Resource Utilisation Daemon and the Resource Availability Daemon, are tightly integrated with the kernel.

6. Performance enhancement strategy

Performance enhancement by a proactive approach involves the identification of the node most capable of handling additional load. This approach has influenced design decisions in information gathering and scheduling.

6.1. Information

Concert is actively engaged in gathering, generating and disseminating information that should be readily available, relevant, accurate and up-to-date. This is achieved by a continual interaction between the information policy modules and the environment. This information is expressed in terms of a load index which is a metric against which the load at each node in the system is measured. Many different load indices were used in load sharing schemes. In Condor [13], for instance, a workstation is idle if no user activity is detected over a short interval. Stealth [14] uses CPU utilization as its load index. Utopia [12], on the other hand, makes use of a wider information base. It consists of load vector, which includes CPU queue length, free memory, disk transfer, swap space and concurrent users. In Concert a similar approach to information gathering was adopted for its scheduling. An investigation of several load measuring methods for various resources was conducted as part of the search for a suitable load index [15]. In Concert the information covers many aspects:

- Information base. The choice and use of a resource availability index instead of a load index reflects the new emphasis that proactivity and performance entail. The index consists of three components: CPU queue availability, CPU utilization availability and primary memory availability. The Resource Availability Daemon is responsible for recording this information. In addition, Concert provides support for predictive models, in common with other schemes such as History [16] and Stealth [14], by keeping a record of resource utilization and behaviour of tasks. The Resource Utilisation Daemon generates information about real and actual processing times, I/O activity, memory usage, process creation and scheduling.
- Local heterogeneity. Accuracy of information is also required when performance heterogeneity occurs in a network, namely when nodes have the same architecture but different levels of resource. The components of the resource availability index are scaled down before dissemination. This normalization process contributes to the availability of accurate information.
- Information update. Scheduling decisions based on obsolete information may affect overall performance. Time-stamping on receipt of messages is used for dealing with the obsolescence of information about resource availability. Resource utilization and task behaviour are updated each time a task executes.
- Information access. The replication of information enables each node to participate actively in the load sharing scheme, and enhances the fault-tolerance of the scheme. Nodes can access local and remote information, and can make scheduling decisions autonomously. They contribute autonomously to the generation of information and to the execution of tasks.

6.2. Scheduling

The implication of the proactive approach is that the scheduling policy should identify the node that is most capable of handling additional load, rather than the node with the least load. The global scheduling policy depends on the resource availability information and the resource utilization information, by selecting the tasks to be migrated and the nodes to which they will be transferred. This policy ensures that tasks are placed at the most appropriate node. Experimental results support this approach and show that keeping more tasks locally lead to better system-wide performance [16].

The scheduling process is triggered either by the entry of a new task into the system or by the activation of an existing task. Only the Distributed Scheduler is involved since the information base, for existing tasks at least, is already set up by the information policy. Two conditions need to be fulfilled before a load transfer is considered:

1. The existence of a significant resource availability differential between nodes to ensure that a load transfer will actually be beneficial.
2. The availability of some threshold amount of resource at the target node.

6.3. Performance

The implementation of a proactive strategy and a distributed control in Concert require replication of information at node level, which may lead to a high volume of traffic across the network. Furthermore, its emphasis on a wide information base may involve processing overheads. It is however the case that wider, readily available, relevant and up-to-date information can lead to quicker and more accurate scheduling decisions. Moreover, the range of information provided by the modules can enhance the flexibility of the policies that can be adopted in a load sharing scheme, as was observed in Chorus [17].

The awareness of the potential inefficiencies at node level has shaped the implementation of the load sharing scheme, in its structure and its interaction mechanism:

- Scheduling policy implementation. As the transfer policy was redundant in a proactive approach, the selection and location policies were merged into one single module, the Distributed Scheduler.
- Kernel modification. A selective modification of the kernel was required for an efficient implementation of the information modules.
- Load metric generation. In order to minimize the costs of generation and dissemination of information and prevent its obsolescence a metric-generation period of ten seconds was chosen, as a compromise.
- Scheduling. The replication of information leads to quicker scheduling decisions. Global scheduling decisions can be made at the node level by the Distributed Scheduler without the need to interact with other nodes.

At the network level the reduction of communication traffic between nodes was motivated by the introduction of various schemes:

- Task migration. Transfer of tasks at initiation time is inherently efficient, compared to those schemes that implement preemptive policies.
- Information dissemination. A state-change policy was adopted in order to reduce the exchange of unnecessary messages. A minimum period between transmissions limits the effect of frequent state changes.
- Network traffic is minimised by the choice of a connectionless protocol, UDP instead of TCP. The

request-reply model provides some form of acknowledgement.

In Concert, the module components of the load sharing scheme were found to be inactive most of the time. Experiments with four nodes showed that the average processing overhead of the load sharing scheme was around 2.83%.

7. Conclusion

In this paper, proactivity in load sharing was presented as a means of achieving performance enhancement in service provision. The emphasis is put on avoiding load imbalance rather than reacting to its detection. A load sharing scheme, Concert, was presented as an illustration of a proactive approach to load imbalance. The distributed control of the scheme and the autonomy of the nodes provide support for reliability and robustness. This scheme offers full transparency to users and tasks, and partial transparency to operating system.

Although this work was confined to the local level within grid computing, future work requires the investigation of the integration of load sharing with grid computing services. The focus of further work will be on the design and implementation of a suitable interface to the grid. Apart form the need to advertise widely the characteristics of the resources available locally, the system should be able to provide a negotiation mechanism as a fundamental component of the scheduling and resource management function.

8. References

[1] Forster I. and Kesselman C. (Eds.), *The Grid: Blueprint for a new Computing Infrastructure*, Morgan Kaufman, 1999.

[2] Skillicorn D.B., Motivating Computational Grids, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2002, Berlin, 401-406.

[3] Abramson D., Buyya R. and Giddy J., A Computational economy for grid computing and its implementation in the Nimrod-G resource broker, *Future Generation Computer Systems*, 18 (2002) 1061-1074.

[4] Czajkowski, K., Fitzgerald, S., Forster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing, Proc. 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), 2001.

[5] Chao K-M., Anane R., Chen and Gatward R., Negotiating Agents in a Market-oriented Grid, Proc. of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2002, 436-437.

[6] Tripath, A., Challenge designing Next generation Middleware systems, CACM 45(6), June 2002.

[7] Milojici, D.S., Douglis, F., Paindeveine, Y., Wheeler, R. and Zhou, S., Process Migration, ACM Computing Surveys, Vol 32, No 3, September 2000, 241-299.

[8] Hao Y., Liu J. S., Kim J., An All-Sharing load-Balancing Scheme on the CSMA/CD Network and its analysis, *The Computer Journal*, 37(4), 1994, 779-794.

[9] Le P., Srinivasan B., A migration tool to support resource and load sharing in heterogeneous computing environments, *Computer Communications*, 20, 1997, 361-375

[10] Thomas A., Neilsen M., A Load Sharing System for a Network of Independent Workstations, *Proc Intl Conf on Intelligent Information Systems*, 1995, 97-100, ISMM-ACTA Press.

[11] Douglis F., Ousterhout J., Transparent Process Migration: Design Alternatives and the Sprite Implementation, *Software - Practice and Experience*, 21(8), 1991, 757-785.

[12] Zhou S., Zheng X, Wang J., Delisle P., Utopia: A Load Sharing Facility for Large Heterogeneous Distributed Computer Systems, *Software - Practice and Experience*, 23(12), 1993, 1305-1336.

[13] Litzkow M. J., Livny M., Mutka M. W., Condor - A Hunter of Idle Workstations, Proc 8th IEEE International Conference on Distributed Computing Systems, June 1998, 104-111.

[14] Krueger P., Chawla R. The Stealth Distributed Scheduler Proceedings of the 11th IEEE International Conference on Distributed Computing Systems, 1991, 36-343.

[15] Anthony R., Goodeve D., A New Metric for expressing CPU Load, Proc 4th International Conference on Computer Science and Informatics, Vol 3, October 1998, 229-234, North Carolina, Association for Intelligent Machinery.

[16] Svensson A., History, an Intelligent Load Sharing Filter, 10th IEEE International Conference on Distributed Computing Systems, 1990, 546-533.

[17] O'Connor M. Tangney B., Cahill V., Harris N., Micro-Kernel support for migration, Distributed Systems Engineering, 1, 1994, 212-223, BCS.