Load Balancing in Distributed Simulations on the Grid

Ming Jiang
School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK
mzj@cs.bham.ac.uk

Rachid Anane
School of Mathematical &
Information Sciences
Coventry University
Coventry, CV1 5FB UK
r.anane@coventry.ac.uk

Georgios Theodoropoulos
School of Computer Science
University of Birmingham
Birmingham, B15 2TT, UK
gkt@cs.bham.ac.uk

Abstract - *Multiprocessors, networks of workstations and cluster computers have well served the computational needs of the simulation community by supporting the distributed simulations of larger models. However, the application of simulation to more complex problems calls for ever more computational power. The emergence of the Grid provides an unrivalled opportunity to address this problem and thus facilitate advances in the modelling and the analysis of large-scale systems by harnessing the power of many computers. The aim of this paper is to present a framework for the execution of optimistic distributed discrete event simulations on the Grid. A generic architecture is described and a load-balancing scheme is introduced. In the context of a Grid-aware TWarp simulation kernel.*

Keywords: Grid computing, distributed simulation, load balancing.

1 Introduction

The advent of Grid computing presents simulation modellers with many opportunities as well as challenges. The seamless access to computational resources enhances and widens the scope of distributed simulation. It provides an answer to the need for powerful machines to support distributed simulation and thus the modelling and analysis of large-scale systems such as environment systems, flexible manufacturing systems and telecommunication systems.

Grids are characterised by the distributed nature of the resources, the lack of centralisation and the transient configuration in resource allocation, and this increasingly, within a highly competitive environment. Although favourable to the execution of computationally intensive simulations, this highly dynamic feature may invalidate the gains that parallelism is expected to yield if the computational load on the nodes is unbalanced.

Load balancing has been the subject of intensive study at the level of a network of workstations (NOWs), both at the operating system level and the application level, as in the case of distributed simulation. Grid computing, however, presents new challenges, not least because of the increased network latency. This aspect precludes the application of traditional load balancing to Grid computing. The aim of this paper is to present a load balancing algorithm for distributed simulations adapted to Grid computing, within a specific framework.

0-7803-8566-7/04/$20.00 © 2004 IEEE.

2 Grid Computing

The level and intensity of the research devoted to Grids and Grid computing is a testimony to their increasing importance. 'Grids are persistent environments that enable software applications to integrate instruments, displays, computational and information resources that are managed by diverse organizations in widespread locations' [4]. Grids are large, dynamic, distributed and heterogeneous resource systems [15], and Grid computing aims at providing transparent access to scarce resources. Different types of Grid have been identified; among them computational Grids are the subject of much research and attention. They are concerned with the efficient execution of tasks on a set of compute servers. In common with other types of Grid, the use and integration of resources in Grid computing involves a set of fundamental middleware services: resource advertisement and discovery, scheduling and transfer of tasks [1].

In advertising its resources a compute resource provider might be required to include the number of nodes in a cluster, the amount of memory, the operating system version and the load average. Resource discovery by a resource requestor is often achieved by means of a directory service or broker/matchmaker. The main function of this service is to locate remote resources and to help route computational requests to the most suitable computer 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 instantaneous load. This service is usually provided by a global scheduler, and the transfer of tasks requires commitment to service provision. Service provision may involve negotiation as an integral part of scheduling.

The ubiquitous nature of the Grids is undoubtedly enhanced by the development and introduction of middleware services. The Globus Toolkit is one significant manifestation of the drive towards providing seamless

access to remote and scarce resources. It is an open, middleware architecture, which offers a rich set of services and libraries that facilitate connectivity and mediation to Grid resources [7]. The Grid services provided by Globus include information bases, resource management, data management, communication, security and fault detection.

3 Distributed Simulation

The term distributed simulation refers to the execution of discrete event simulation models on parallel/distributed platforms [5]. Various approaches for exploiting parallelism at different levels have been followed. Decentralised, event-driven simulation has the greatest potential for high performance and consequently, has attracted considerable attention from the research community and has almost exclusively been employed for practical simulation applications. Following this approach, a simulation consists of a number of parallel logical processes (LPs), each operating independently and communicating with its peers to exchange information. Each LP maintains a local clock with the current value of the simulated time, Local Virtual Time (LVT). This value represents the process's local view of the global simulated time and denotes how far in simulated time the corresponding process has progressed. An LP will repeatedly accept and process messages arriving on its input links, advancing its LVT and possibly generating, as a result, a number of messages on its output links. The timestamp of an output message is the LVT of the LP when the message was sent.

A fundamental problem in event-driven distributed simulation is to ensure that the LPs always process messages in increasing timestamp order, and hence faithfully and accurately implement the causal dependencies and partial ordering of events dictated by the causality principle in the modelled system. Synchronous approaches utilise global synchronisation schemes (implemented in a centralised or decentralised fashion) to force the LPs to advance together in lock step. This guarantees that the causality principle is not violated but the potential for speedup is limited. In contrast, in asynchronous simulation, LPs operate asynchronously, advancing at completely different rates, simultaneously processing events which occur at completely different simulated times. This approach has greater potential for speedup, but additional synchronisation mechanisms are required to ensure that the LPs adhere to the *local causality constraint* and process messages in increasing timestamp order.

Two main approaches have been developed to ensure that the local causality constraint is not violated in asynchronous simulation, namely *conservative* and *optimistic*. Conservative approaches strictly avoid causality errors but can introduce deadlock problems. In addition, conservative techniques rely heavily on the concept of lookahead, and are thus typically suitable only for applications with good lookahead properties. Conservative protocols also typically require a static partition and configuration of the distributed model, and systems with dynamic behaviour are in general difficult to model. Optimistic approaches allow the processes to advance optimistically in simulated time, detecting and recovering from causality errors by means of a *rollback* mechanism which forces processes to undo past operations. For the rollback of the simulation to be feasible, each process must hold information regarding its recent history (e.g., previous state vectors, processed input events, and previously sent output messages) up to last 'correct time', referred to as the *Global Virtual Time* (GVT). GVT is generally the smallest local clock value amongst all the LPs, and is periodically computed and distributed to all the logical processes. In contrast to conservative approaches, optimistic approaches can accommodate the dynamic creation of logical processes and do not require the prediction of future events for their efficient operation.

Efficient management of resources, such as CPU and memory, can have a direct effect on the synchronization overhead and thus plays a crucial role in the performance of distributed simulations. For instance, in an optimistic simulation, the amount of memory available can define the degree of optimism of a Logical Process and the amount of rollbacked operations.

4 Load Balancing Distributed Simulations on a Grid

The synchronisation mechanisms involved in distributed simulation render conventional load balancing techniques insufficient for this class of parallel applications. For instance, in the case of optimistic synchronisation, high processor utilisation does not necessarily imply good performance as operations could later be undone(rollback), while process migration can affect the efficiency of the synchronisation mechanism (e.g. amount of roll backed computation). Consequently, the load balancing has been studied extensively in the context of both conservative and optimistic parallel simulation and several load balancing lagorithms have been developed, e.g. [2][3][6][8][13][14][16].

Most, if not all, of these algorithms assume parallel/distributed platforms with negligible interprocessor commmunication delays (such as multiprocessors, networks of workstations and cluster computers). Thus their feedback loop is based mainly on processor load and they largely tend to ignore communication latencies.

In a Grid, however, communication delays are substantial and can be the decisive factor of the simulation performance. This issue is increasingly receiving the attention on researchers in the field. In [11] and [12], Iskra

et al. investigate cancellation strategies and message aggregation techniques for optimistic distributed simulations on a Grid. In [10] Mikler, Das and Fabbri propose mechanisms that allow LPs to tune their resource demands and optimism as a function of the undelying network performance. In this paper we look at another related problem, namely load balancing. In particular, we investigate how communication latencies can be taken into account for the dynamic migration of LPs (or state) between remode nodes (e.g. clusters) in a Grid.

As a first step in our endeavour, we have decided to utilise an existing algorithm and investigate how it can be extended to deal with the communication latencies in a Grid. The base algorithm that we have chosen was developed by Carothers and Fujimoto [3], and is part of the standard distribution of the GTW optimistic kernel [9]. This appears to be an appropriate choice as the algorithm was originally developed for the execution of optimistic simulations on multi-user, heterogeneous networks of workstations (NoWs) and takes into account both dynamic availability of resources (processors) and load of other users, typical features of a Grid environment too.

A detailed description of the algorithm is outside the scope of this paper and it can be found in [3]. Briefly, the algorithm first decides dynamically on a usable set of processors, namely processors that are not too oberloaded and therefore can be used for the simulation (the *processor allocation* policy) and then employs a *load balancing* policy to migrate LPs from overloaded processors to less loaded ones. The algorithm uses two central metrics : The processor advance time (PAT) and the cluster (of LPs) advance time (CAT) the amount of wall clock time required for a processor to advance one unit of simulated time in the absence of rollback. Thus PAT represents the load of a processor : the higher the PAT the more loaded the processor. The load balancing algorithm first sorts the usable set of processors based on their PAT values and then traverses the sorted set, migrating (clusters of) LPs from processors with large PAT values to those with lower PAT values so that the maximum difference between the PAT values in any two processors is minimised. Which clusters to migrate, is decided by CAT, which defines the amount of computation required to advance a cluster one unit of simulated time in the absence of rollback. A large CAT value implies high computational requirements and therefore clusters with high CAT are give priority in the migration.

4.1 A Grid-aware Load Balancing Algorithm

The algorithm described above clearly does not take communication latencies into account. Assuming a platform where a number of (processor) clusters are connected together in a Grid, the above algorithm could migrate an LP to a distant processor with a low PAT (fast), even though this LP may communicate more frequently with local LPs, thus increasing the communication cost (since now the messages need to be sent over a high latency link) possibly nullifying the performance gains achieved by utilising the faster processor.

The above scenario leads to an important observation, namely that any load balancing and migration policy for the Grid should exploit locality and take into account the communication patterns between the LPs and aim to place apart at distant nodes LPs which do not communicate frequently.

To extend the algorithm, we define two values : Processor-to-Processor Communication (PPC) and Cluster-to-Processor Communication (CPC). We assume that a centralised agent exists, which monitors the simulation and collects information regarding the communication patterns and the messages exchanged between the LPs. PPC and CPC indicate the total number of messages exchanged between two processors (between any LPs) and between a cluster and a processor (any LP on that processor) respectively within a time interval.

Based on the above values, our algorithm proposes to migrate load from a high PAT processor P_i to a lower PAT processor P_j based on the following formula :

$$PCSij = \frac{a \times PPCij \times LATENCYij}{b \times PATj} \quad (1)$$

where PCS_{ij} is the Processor-Communication-Speed for processor P_j and for the link from Pi to P_j . During the migration phase, load from P_j will be moved to the processor P_i with the highest PCS_{ij}. In other words, the load will migrate to a procesor that is not overloaded (low PAT), is *far* (i.e. with high communication latency) and has exhibited a high level of communication with the source processor. The variables a and b are parameters that define the relative importance of processor and communication speed and can be tuned for a particular Grid configuration. For new processors that become available, their PPC is initially set to the average PPC value for all the other processors in the usable set. The cluster of LPs to migrate from processor P_i to Processor P_j will be that with the highest CAT*CPC, namely the one with the highest computational requirements and the highest contribution to the communication traffic between the processors.

Figures 1 and 2 illustrate the expanded algorithm, highlighting the modifications done to the original algorithm as explained above.

```
Function DynamicLoadManagementModule()
{
 /* Processor Allocation Policy */
 For Each Processor, PE[i]
    If (PE[i] Is ACTIVE AND PE[i]'s Load Average > DeAllocThreshold)
      Set PE[i] to INACTIVE;
      UnLoad ALL LP Clusters from PE[i] to an ACTIVE Neighbouring Processor;
      Set PE[i]'s PAT to INFINITY;
      NumActiveProcessor--;
    endif
    If (PE[i] Is INACTIVE AND PE[i]'s Load Average < AllocThreshold)
      Set PE[i] to ACTIVE;
      Set PE[i]'s PAT to MINI_PAT;
      NumActiveProcessor++;
    endif

 /* Load Balancing Policy */
Compute PAT value for each ACTIVE processor; /* PAT is the Processor Advance Time, which
indicates the amount of wall-clock time required for a processor to advance one unit of simulated time
in the absence of rollback. */

 Compute the mean value, m, and standard deviation value, d, over PATs of all ACTIVE processors;

 The number of migration is set to 0;

 Done=FALSE;

 While(! Done)
    {
     For Each ACTIVE Processor, PE[i]
      {
       If (PE[i]'s PAT < m-Kd)
          Put PE[i] into the fast progress processor set FP;
       If (PE[i]'s PAT > m+Kd)
          Put PE[i] into the slow progress processor set SP;
      }/* K>0 and is a user pre-defined parameter before the execution of simulation. */
    If ((FP∪SP = = ∅) OR (the number of migration = = MAX_MIGRATION_NUM))
      Done=TRUE;
    Else
      {
       Select the processor A, with the MAX. PAT value in SP set as the donation processor;
       Sort the rest of the processors in FP according to their ( a × PPC × LATENCY / b × PAT ) value;
/* PPC is the level of Processor-to-Processor-Communication during last interval T and here PPC
value is about the processors in FP to processor A. For each new added ACTIVE processor, its PPC
value is set to the average value of other PPCs. LATENCY is the Grid network latency between
processor A and the processors in FP. Parameters a and b are application and Grid environment
depended control parameters, which are decided by experiments. */

       If (TryToOffLoad()) = = FAILED )
         Done=TRUE;
      }
    The number of migration increases by 1;
    }
}
```

Figure 1 : The Extended Load Balancing Algorithm (Part A)

```
Function TryToOffLoad()
{
 For each processor, B, in FP set
  {
    Sort all LP clusters on processor A according to their (CAT × CPC) values;
/* CAT is the LP Cluster Advance Time, which indicates the amount of computation time required to advance one unit
simulation time in the absence of rollback. CPC  is the level of Cluster-to-Processor-Communication during last
interval T and here CPC is about cluster c to processor B. If B is a new added ACTIVE processor, the CPC of cluster
c is set to 1. */
    For each LP cluster, c, assigned to processor A
      {
       If (moving cluster, c, from A to B reduces the difference in PAT values between A and B)
        {
          Move Cluster c from A to B;
          Update the PAT values for processor A and B;
          return(SUCCEEDED);
        }
      }
  }
 return(FAILDED);
```

Figure 2 The Extended Load Balancing Algorithm (Part B)

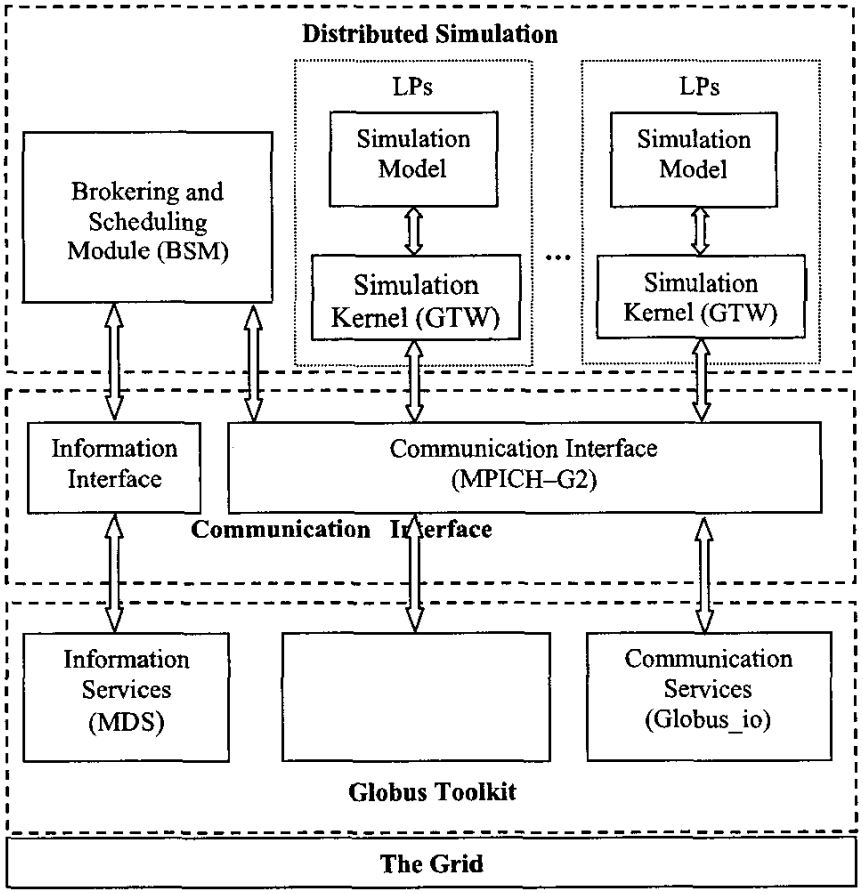

Figure 3 : The Simulation Framework

5 A Simulation Framework

The investigation of load balancing algorithms is an integral part of a research programme aimed at designing and developing an architectural framework for distributed simulation on a Grid. To this end a proposed architecture hase been proposed as an implementation of the framework (Figure 3) [17] . It consists of four fundamental layers: the distributed simulation, the communication, the middleware and the Grid layers.

The two lower layers are closed linked since the Grid is mediated by the Globus toolkit, which provides services for information discovery (MDS), scheduling (GRAM) and for basic input/output operations and communication (Globus_io).

The communication layer plays a pivotal role between the Grid middleware services and the modules that make up the distributed simulation layer, and thus ensuring interoperability between the layers. For this purpose a Grid aware communication mechanism (MPICH-G2) was utilised because of its ability to interoperate with the Globus toolkit, on the lower layer. On the above layer, it is the GTW optimistic distributed simulation kernel that was enhanced to make it aware of MPICH-G2.

The distributed simulation layer incorporates two main components, the brokering and scheduling module (BSM) and a set of logical processes (LPs) distributed over the Grid. In the framework, the BSM has an indirect information interface, in the communication layer, to the MDS of Globus. This layer presents a highly dynamic set of entities, especially the logical processes.

Most of the components on the different layers have been integrated into a coherent system that supports distributed simulation. The system operates at this stage without the DSM module. A number of distributed simulation experiments have been conducted successfully on the framework. The focus of the work at this stage, however, is on load balancing as an optimising technique in distributed simulation. Load balancing is seen as a subsidiary function for the scheduling process performed by the BSM, and depends on the discovery and matching functions of the brokering service.

6 Conclusion

This paper has presented an architectural framework for supporting distributed simulation on a Grid. One of the key issues of the framework is the efficient management of resources, which is crucial for distributed simulations. The paper has introduced an adaptation of an algorithm for dynamic load balancing of optimistic simulations to Grid environments. The performance of the framework has been partially evaluated by executing typical benchmark distributed simulations on a Globus-enabled network of workstations running Linux. Work is being carried out to incorporate an implementation of the load balancing algorithm into the brokering and scheduling module (BSM). Future work will focus on the evaluation of the enhanced framework by the inclusion of the BSM and the load balancing scheme in particular.

References

[1] 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), pp1061-1074.

[2] C. Burdorf and J. Marti, "Load balancing strategies for Time Warp on multi-user workstations," The Computer Journal, vol. 36, no. 2, pp. 168– 176, 1993.

[3] C. Carothers and R. Fujimoto, "Background execution of Time-Warp programs," in Proceedings of 10th Workshop on Parallel and Distributed Simulation. Society for Computer Simulation, 1996, Society for Computer Simulation.

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

[5] R. M. Fujimoto. Parallel and Distributed Simulation Systems. John Wiley & Sons, Inc., New York, 2000.

[6] D. W. Glazer and C. Tropper, "On process migration and load balancing in Time-Warp," IEEE Transactions on Parallel and Distributed Systems, vol. 3, no. 4, pp. 318–327, 1993

[7] The Globus Project, http://www.globus.org/

[8] A. Goldberg, "Virtual time synchronisation of replicated processes," in Proceedings of 6th Workshop on Parallel and Distributed Simulation. Society for Computer Simulation, 1992, pp. 107–116, Society for Computer Simulation.

[9] The GTW Homepage, http://www.cc.gatech.edu/ computing/pads/teddoc. html

[10] A. Mikler, S. Das and A, Fabbri, "Distributed Simulation for Large Communication Infrastructures Across Loosely Coupled Domains", Proceedings of the 6th International Conference on Telecommunication Systems, Modelling and Analysis, Nashville, Tennessee (USA), March 1998.

[11] K. Iskra, G. Albada and P. Sloot, "Time Warp Cancellations Optimisations on High Latency Networks",

in Proceedings of the 7th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'03), 2003.

[12] K. Iskra, G. Albada and P. Sloot, "Towards a Grid-aware Time Warp", In in Proceedings of the Workshop on Parallel and Distributed Simulation. Society for Computer Simulation, 2004 (PADS'2004), Kufstein, Austria, pp. 63-70

[13] P. L. Reiher and D. Jefferson, "Dynamic load management in the Time- Warp operating system," Transactions of the Society for Computer Simulation, vol. 7, no. 2, pp. 91–120, 1990.

[14] R. Schlagenhaft, M. Ruhwandl, C. Sporrer, and H. Bauer, "Dynamic load balancing of a multi-cluster simulation on a network of workstations," in Proceedings of 9th Workshop on Parallel and Distributed Simulation. Society for Computer Simulation, 1995, pp. 175–180, Society for Computer Simulation.

[15] Skillicorn D.B., Motivating Computational Grids, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, May 2002, pp401-406.

[16] Theodoropoulos, G., Logan, B., "Interest Management and Dynamic Load Balancing in Distributed Simulation" Proceedings of 12th European Simulation Symposium (ESS'2000), 28-30 September 2000, University of Hamburg, Hamburg, Germany, pp. 111-115, Society for Computer Simulation International, ISBN 1-56555-190-7, editors: Dietmar Moeller

[17] M. Jiang, G. Theodoropoulos. R. Anane, "A Framework for Distributed Simulation on a Grid", Winter International Symposium on Information and Communication Technologies, ACM WISICT04, Hyatt Regency, Cancun, Mexico, January 5-8th, 2004