Experimental Studies in Load Balancing[*]

Azzedine Boukerche and Sajal K. Das

Department of Computer Sciences, University of North Texas, Denton, TX. USA

Abstract. This paper takes an experimental approach to the load balancing problem for parallel simulation applications. In particular, it focuses upon a conservative synchronization protocol, by making use of an optimized version of Chandy&Misra null message method, and propose a dynamic load balancing algorithm which assumes no compile time knowledge about the workload parameters. The proposed scheme is also implemented on an Intel Paragon A4 multicomputer, and the performance results for several simulation models are reported.

1 Introduction

A considerable number of research projects on load balancing in parallel and distributed systems in general has been carried out in the literature due to the potential performance gain from this service. However, all of these algorithms are not suitable for parallel simulation since the synchronization constraints exacerbate the dependencies between the LPs. In this paper, we consider the load balancing problem associated with the conservative synchronization protocol that makes use of an optimized version of Chandy-Misra Null messages protocol [3]. Our primary goal is to minimize the synchronization overhead in conservative parallel simulation, and significantly reduce the total execution time of the simulation by evenly distributing the workload among processors [1, 2].

2 Load Balancing Algorithm

Before the load balancing algorithm can determine the new assignment of processes, it must receive information from the processes. We propose to implement the load balancing facility as two kinds of processes: *load balancer* and *process migration* processes. A load balancer makes decision on *when* to move *which* process to *where*, while migration process carries out the decision made by the load balancer to move processes between processors.

To prevent the bottleneck and reduce the message traffic over a fully distributed approach. we propose a Centralized approach (CL), and a Multi-Level (ML) hierarchical scheme; where processors are grouped and work loads of processors are balanced hierarchically through multiple levels. At the present time,

[*] Part of this research is supported by Texas Advanced Technology Program grant TATP-003594031

we settle with a Two-Level scheme. In level 1, we use a centralized approach that makes use of a *(Global) Load Balancing Facitilty, (GLBF)*, and the global state consisting of process/processors mapping table. The motivation for this is that many contemporary parallel computers are usually equipped with a front-end host processor (e.g., hypercube multiprocessor machines). The load balancer ($GLBF$) sends periodically a message $<$ *Request_Load* $>$ to a specific processor, called $first\_proc$, within each clusters requesting the average load of each processor within each groups.

In level 2, the processors are partitioned into clusters, and the processors within each group are structured as a virtual ring, and operate in parallel to collect the work loads of processors. A virtual ring is designed to be traversed by a token which originates at a particular processor, called *first_proc*, passes through intermediate processors, and ends it traversal at a preidentified processor called *last_proc*. Each of the ring will have its own circulating token, so that information (i.e., work loads) gathering within the rings is concurrent. As the token travels through the processors of the ring, it accumulates the information (the load of each processors and *id* of the processors that contain the highest/lowest loads), so that when it arrives at the *last_proc*, information have been gathering from all processors of the ring. When all processors have responded, the load balancer ($GLBF$) computes the new load balance. Once the load balancer makes the decision on *when to move which LP to where*, it sends $Migrate_{Request}$ to the migration process which in turns initiates the migration mechanism. Then, the migration process sends the (heavily overloaded) selected process to the lighted underloaded neighbor processor.

The computational load in our parallel simulation model consists of executing the null and the real messages. We wish to distribute the null messages and the real messages among all available processors. We define the (normalized) Load at each processor (Pr_k) as follows $Load_k = \mathcal{F}(R^k_{avg}, N^k_{avg}) = \alpha R^k_{avg}/R_{avg} + (1 - \alpha)N^k_{avg}/N_{avg}$; where R^k_{avg},and N^k_{avg} are respectively the average CPU-queue length for real messages and null messages at each processor Pr_k; and α and (1 - α) are respectively the relative weights of the corresponding parameters. The value of α was determined empirically.

3 Simulation Experiments

The experiments have been conducted on a 72 nodes Intel Paragon, and we examined a general distributed communication model (DCM), see Fig. 1, modeling a national network consisting of four regions where the subnetworks are a toroid, a circular loops, a fully connected, and a pipeline networks. These regions are connected through four centrally located delays. A node in the network is represented by a process. Final message destinations are not known when a message is createad. Hence, no routing algorithm is simulated. Instead one third of arrival messages are forwarded to the neighboring nodes. A uniform distribution is employed to select which neigbor receives a message. Consequently the volumes of messages between any two nodes is inversely proportional to their hop distance.

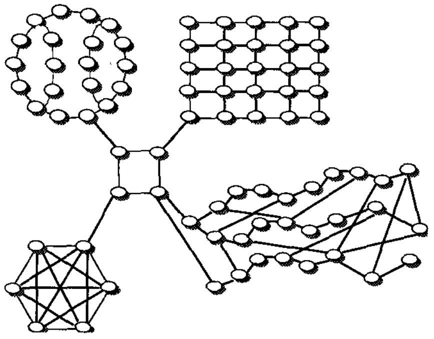

Fig. 1. Distributed Communication Model

Messages may flow between any two nodes and typically several paths may be employed. Nodes receive messages at varying rates that are dependant on traffic patterns. Since there are numerous deadlocks in this model, it provides a stress test for any conservative synchronization mechanisms. In fact, deadlocks occurs so frequently and null messages are generated so often that the load balancing strategy is almost a necessity in this model.

Various simulation conditions were created by mimicking the daily load fluctuation found in large communication network operating across various time zones in the nation [1]. In the pipline region, for instance, we arranged the subregion into stages, and all processes in the same stages perform the same normal distribution with a standard deviation 25%. The time required to execute a message is significantly larger than the time to raise an event message in the next stage (message passing delay).

The experimental data was obtained by averaging several trial runs. The execution time (in seconds) as function of the number of processors are presented in the form of graphs. We also report the *synchronization overhead* in the form of null message ratio (NMR) which is defined as the number of null messages processed by the simulation using Chandy-Misra null-message approach divided by the number of real messages processed. the performance of our dynamic load balancing algorithms were compared with a static partitioning.

Figure 2 depicts the results obtained for the DCM model. We observe that the load balancing algorithm improves the running time. In other words, the running time for both network models, ie., Fully Connected and Distributed Communication, decreases as we increase the number of processors. The results in Fig. 6 indicate a reduction of 50% in the running time using the CL dynamic load balancing strategy compared with the static one when the number of processors is increased from 32 to 64, and about 55-60% reduction when we use the ML dynamic load balancing algorithm.

Figure 3 displays NMR as a function of the number of processors employed in the network models. We observe a significant reduction of null-messages for all load balancing schemes. The results show that NMR increases as the number of processors increases for both population levels. For instance, If we confine ourselves to less than 4 processors, we see approximately 30-35% reduction of

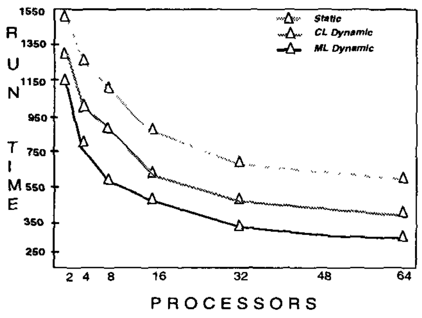

Fig. 2. Run Time Vs. Nbr of Processors

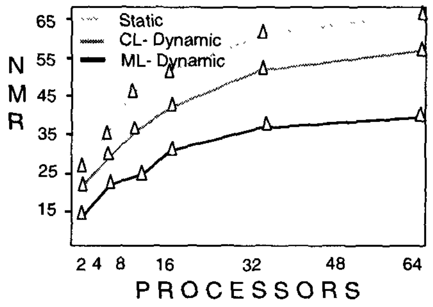

Fig. 3. NMR Vs. Nbr of Processors

the synchronization overhead using the CL dynamic load balancing algorithm over the static one. Increasing the number of processors from 4 to 16, we observe about 35-40% reduction of NMR. We also observe a 45% reduction using CL strategy over the static one, when 64 processors. are used, and a reduction of more than 50% using ML scheme over the static one.

These results conclude that the multi-level load balancing strategy significantly reduces the synchronization overhead when compared to a centralized method. In other words, a careful dynamic load balancing improves the performance of a conservative parallel simulation.

4 Conclusion and Future Work

We have described a dynamic load balancing algorithm based upon a notion of CPU-queue length utilization, and in which process migration takes place between physical processors. The synchronization protocol makes use of Chandy-Misra null message approach. Our results indicate that careful load balancing can be used to further improve the performance of Chandy-Misra's approach. We note that a Multi-Level approach seems to be a promising solution in reducing further the execution time of the simulation. An improvement between 30% and 50% in the event throughput were also noted which resulted in a significant reduction in the running time of the simulation models.

References

1. Boukerche, A., Das, S. K.: Dynamic Load Balancing Strategies For Parallel Simulations. TR-98. University of North Texas.
2. Boukerche, A., and Tropper C., "A Static Partitioning and Mapping Algorithm for Conservative Parallel Simulations", IEEE/ACM PADS'94, 1994, 164–172.
3. Misra, J., "Distributed Discrete-Event Simulation", ACM *Computing Surveys*, Vol. 18, No. 1, 1986, 39–65.