Parallel Job Scheduling Policy for Workstation Cluster Environments
J. H. Abawajy and S. P. Dandamudi
Centre for Parallel and Distributed Computing, School of Computer Science, Carleton University,
Ottawa, Ontario, K1S 5B6, Canada.
{abawjem, sivarama}@scs.carleton.ca

1 Introduction

As workstation clusters (WC) become more commonly used for parallel jobs, there is a growing awareness for the need of job scheduling policies. There have been a fair number of studies on how to schedule parallel applications on parallel systems and a good survey in the area can be found in [5]. It has been shown that the best solution to the processor allocation problem in a distributed multiprocessor environment is an adaptive scheduling policy that can adjust load distribution based on runtime scheduling algorithms [2,3]. The main idea of adaptive space-sharing policies is that the number of processors assigned to a job is a compromise between the user’s request and what the system can provide. Note that there are differences in the architecture of the multiprocessor systems and WC-based distributed systems. For example, the processors in the multiprocessors systems are usually homogenous whereas those of WC are usually heterogeneous. This change of architectural environment requires important differences in the decisions made by the system scheduling policy. Most of adaptive scheduling policies for WC-based systems provide only rudimentary facilities for partitioning, i.e., space sharing, the processors among parallel jobs. In addition, parallel applications targeted to WC are typically resource-intensive, i.e. they require more resources than are available at a single site. However, existing adaptive scheduling policies cannot accommodate this requirement. This is because they may assign 1 processor to a job in the extreme cases [2] or lead to a processor fragmentation problem [3], i.e. groups of processors idle within a partition while some jobs may be waiting for free processors, which leads to low system utilisation and throughput.

To address these scheduling issues, we propose a group-based scheduling policy that we refer to as a Cluster-based Adaptive Parallel job Scheduling (CAPS) policy. CAPS is based on Federated Distributed System (FDS) framework [1], a software construct that logically organises and manages a large number of processors by partitioning them into a set of N non-overlapping clusters. Each cluster contains B processors and each processor in the system is belongs to one cluster. CAPS is similar to adaptive policies proposed in [2] and [3] in that they all schedule tasks at run time and their scheduling decisions, in principle, depend on and adapt to the run time system state. However, CAPS is substantially different from the adaptive policies proposed in [2] and in [3] in that it is distributed in the sense that there is no central authority that holds knowledge of and control over when and where different tasks are executed. Instead, the processors determine when and how much work they are willing to do. In addition, CAPS combines the best attributes of the centralised and distributed schedulers while avoiding/minimising their drawbacks. Also, due to its group-based architecture, CAPS is scalable to a large number of workstations.

2 CAPS Policy

CAPS is a two-level scheduling model where the top-level scheduler called a system manager (SM) is responsible for distributing work to the clusters on demand from the later. The bottom level scheduler called cluster manager (CM) runs on each cluster and is responsible for managing and distributing parallel tasks to the processors under its control. Each manager keeps the count of locally running (T_{LR}) and locally waiting (T_{LW}) tasks.

The processors order work from their cluster managers, which in turn orders work from the system manager. When a demand for work arises, it is either satisfied immediately from on-hand inventory, if available, or is backordered otherwise. We say that a demand issued by a node x to a node y is pending if node x has not received a reply from node y. A node with pending demand will not generate another demand until the previous demand is satisfied.

The scheduling algorithm is as follows. Parallel jobs are submitted to the system manager. When a processor P becomes available for work, it orders T_{P} tasks where T_{P} \geq 1 from its CM_{P}. When the CM_{P} receives the work order from the P, if it does not have work to send to P, it will in turn order T_{CM} tasks from the SM, where T_{CM} is computed as follows:

\[ T_{CM} = B \cdot \left( T_{LR} \cdot f \right), \quad 0 < f \leq 1 \quad (I) \]

where B is the number of processors under CM_{P} control and f is a transfer factor. When the SM receives the work order from the CM_{P}, if it does not have work to send to the CM_{P}, the order is backlogged for future processing. Otherwise, a quantity T_{C} of work as determined by the following is dispatched to the CM_{P}:

\[ T_{C} = \begin{cases} \min(T_{W}, T_{CM}) & M = T \\ \min\left(J_{W}, \max\left(1, \left\lfloor \frac{J_{W}}{C} \right\rfloor \right) \right) & O.W. \end{cases} \quad (2) \]

This is the Request/Reply architecture of the multiprocessor systems and WC-based distributed systems. For example, the processors in the architecture of the multiprocessor systems and WC-based distributed systems. For example, the processors in the architecture of the multiprocessor systems and WC-based distributed systems. For example, the processors in the architecture of the multiprocessor systems and WC-based distributed systems. For example, the processors in the architecture of the multiprocessor systems and WC-based distributed systems.
3 Performance Evaluation

The performance of CAPS is compared with two well-known adaptive scheduling policies proposed in [2] and [3] over a wide range of system parameters. We used three workloads, the same as in [4], referred to as W1, W2 and W3 for performance evaluation of the different scheduling policies. A quick description of each workload is as follows. In W1, the service demand of the job is divided evenly among the tasks. In W2, 50% of the tasks of the job receive 50% of the job service demand while the remaining service demand is shared equally amongst the other tasks. In W3, a job’s total service demand such that 50% of the tasks receive 25% of the job service demand while the remaining service demand is shared equally amongst the other tasks.

Figure 1 plots the mean response time of the three policies as a function of offered system load for the three workloads. The result show that CAPS performs, depending on the workload type and the system load, over 50% better than the two adaptive policies. In the experiment, the system load is represented by four parameters. The job arrival process at each node is characterised by a mean inter-arrival time $1/\lambda$ and a coefficient of variation ($CV_a$). Jobs are characterised by a processor service demand (with mean $1/\mu$), degree of parallelism ($P_{\text{max}}$) and a service demand coefficient of variation $CV_s$. The maximum parallelism, $P_{\text{max}}$, of a job is uniformly distributed between 2 and the system size (N). We used the 2-stage hyper-exponential distribution to generate the service time distribution for the jobs. The sequence of job arrivals is generated once and reused for each data point and each scheme. Only the mean inter-arrival time is changed to create different load conditions.

We model communication time to send a message of k words with the linear function $T_{\text{comm}} = T_s + k \cdot T_w$, where $T_s = 728\mu s e c$ and $T_w = 0.92\mu s e c$.

4 Conclusion

This paper presented an algorithm for scheduling large-scale parallel applications in workstation cluster environments. The performance of the proposed scheduling policy is compared with ASP [2] and AP1 [3] through simulation and shown that it performs substantially better than the two adaptive policies.

Acknowledgement: This research is partially supported by Natural Science and Engineering Research Council of Canada and Carleton University.

5 References


Figure 1: Relative performance of the three policies under workload W1 (top), workload W2 (middle), and workload W3 (bottom). The vertical axis plots the mean response time while the horizontal axis plots the system load.