Enhancements to the Decision Process of the Self-Tuning dynP Scheduler

Achim Streit

PC^2- Paderborn Center for Parallel Computing
Paderborn University
33102 Paderborn, Germany
streit@upb.de
http://www.upb.de/pc2

Abstract. The self-tuning dynP scheduler for modern cluster resource management systems switches between different basic scheduling policies dynamically during run time. This allows to react on changing characteristics of the waiting jobs. In this paper we present enhancements to the decision process of the self-tuning dynP scheduler and evaluate their impact on the performance: (i) While doing a self-tuning step a performance metric is needed for ranking the schedules generated by the different basic scheduling policies. This allows different objectives for the self-tuning process, e. g. more user centric by improving the response time, or more owner centric by improving the makespan. (ii) Furthermore, a self-tuning process can be called at different times of the scheduling process: only at times when the characteristics of waiting jobs change (half self-tuning), i. e. new jobs are submitted; or always when the schedule changes (full self-tuning), i. e. when jobs are submitted or running jobs terminate.
We use discrete event simulations to evaluate the achieved performance. As job input for driving the simulations we use original traces from real supercomputer installations. The evaluation of the two enhancements to the decision process of the self-tuning dynP scheduler shows that a good performance is achieved, if the self-tuning metric is the same as the metric used measuring the overall performance at the end of the simulation. Additionally, calling the self-tuning process only when new jobs are submitted, is sufficient in most scenarios and the performance difference to full self-tuning is small.

1 Introduction

Resource management systems (RMS) for modern high performance computing (HPC) clusters consist of many components which are all vital in keeping the machine fully operational. An efficient usage of the machines is important for users and owners, as such systems are rare and high in cost. With regards to performance aspects all components of a modern RMS should perform their assigned tasks efficiently and fast, so that no additional overhead is induced. However, if resource utilization and job response times are addressed, the scheduler plays a major role. A clever scheduling strategy is essential for a high utilization of the

D. Feitelson, L. Rudolph, and U. Schwiegelshohn (Eds.): JSSPP 2004, LNCS 3277, pp. 63–80, 2005.

machine and short response times for the jobs. However, these two objectives are contradicting. Jobs tend to have to wait for execution on a highly utilized system with space sharing. Short or even no waiting times are only achievable with low utilizations or time-sharing. Typically a scheduling policy that optimizes the utilization prefers jobs needing many resources for a long time. Jobs requesting few resources for a short amount of time may have to wait longer until adequate resources are available. If such small and short jobs are preferred by the scheduler, the average waiting time would be reduced. As jobs typically have different sizes and lengths, fragmentation of the schedule occurs and the utilization drops [1]. The task of the scheduler is to find a good compromise between optimizing these two contrary metrics.

Cluster systems usually have a large user community with different resource requirements and general job characteristics [3]. For example, some users primarily submit parallel and long running jobs, whilst others submit hundreds of short and sequential jobs. Furthermore, the arrival patterns vary between specific user groups. Hundreds of jobs for a parameter study might be submitted in one go via a script. Other users might only submit their massively parallel jobs one after the other. This results in a non-uniform workload and job characteristics that permanently change. The job scheduling policy used in a RMS is chosen in order to achieve a good overall performance for the expected workload. Most commonly used is first come first serve (FCFS) combined with backfilling [7,14,9], as on average a good performance for the utilization and response time is achieved. However, with certain job characteristics other scheduling policies might be superior to FCFS. For example, for mostly long running jobs, longest job first (LJF) is beneficial, whilst shortest job first (SJF) is used with mostly short jobs [1]. Hence, a single policy is not enough for an efficient resource management of clusters. Many modern RMSs have several scheduling policies implemented, or it is even possible to replace the scheduling component.

The remainder of this paper is structured as follows. In Section 2 some related work on self-tuning and dynamic policy switching is given. Section 3 starts with a short history of development, contains the concept of the self-tuning dynP scheduler, and presents the different decider mechanisms and enhancements to them. The used performance metric and the applied workload for the evaluation are presented in Section 4. The evaluation in Section 5 starts with a look on the performance of the three basic policies. Then the evaluation results of the different self-tuning metrics and of the comparison of half vs. full self-tuning are presented. The paper ends with conclusions an a brief outlook on future work.

2 Related Work

In [13] the problem of scheduling a machine room of MPP-systems is described. Users either submit long running batch jobs or they work interactively (typically only for a short time). To accomplish this on a single MPP-system the resource management system has to switch from batch mode (preferring batch jobs) to interactive mode (preferring interactive jobs) and back. Usually this is done man-