LOMARC — Lookahead Matchmaking for Multi-resource Coscheduling

Angela C. Sodan and Lei Lan

University of Windsor, Computer Science
401 Sunset Ave., Windsor ON N9B 3P4, Canada
acsodan@cs.uwindsor.ca, lan_lei@hotmail.com

Abstract. Job scheduling typically focuses on the CPU with little work existing to include I/O or memory. Time-shared execution provides the chance to hide I/O and long-communication latencies though potentially creating a memory conflict. We consider two different cases: standard local CPU scheduling and coscheduling on hyperthreaded CPUs. The latter supports coscheduling without any context switches and provides additional options for CPU-internal resource sharing. We present an approach that includes all possible resources into the schedule optimization and improves utilization by coscheduling two jobs if feasible. Our LOMARC approach partially reorders the queue by lookahead to increase the potential to find good matches. In simulations based on the workload model of [12], we have obtained improvements of about 50% in both response times and relative bounded response times on hyperthreaded CPUs (i.e. cut times by half) and of about 25% on standard CPUs for our LOMARC scheduling approach.

1 Introduction

The primary goal in job scheduling is to provide good response times to users. A secondary goal is to improve utilization. Both objectives may conflict with each other though often improved response times also mean improved (though potentially not optimum) utilization. The relationship between them is typically not well expressed. The best response-time behavior so far has been reported for gang scheduling [16,6]. Gang scheduling is a time-sharing approach and means that all processes of the same job are scheduled across nodes at the same time by globally synchronous time slicing [6]. Gang scheduling has shortcomings as regards latency hiding for I/O and long-latency communication. Latency hiding plays an increasingly significant role for the emerging class of data-intensive applications like datamining. Long-latency hiding is important also for potential grid applications. Loosely coordinated coscheduling [1,24,17,34,27] (avoiding the globally synchronous execution and enabling to release the CPU if waiting) and relaxed combinations of gang and local CPU scheduling [23] provide alternatives performing better in this regard. Loosely coordinated coscheduling requires modifications of the communication software and potentially the OS, typically using a spin-blocking approach to release the CPU after some time

D. Feitelson, L. Rudolph, and U. Schwiegelshohn (Eds.): JSSPP 2004, LNCS 3277, pp. 288–315, 2005.

of waiting and a priority boost to schedule processes that have been waiting for communication. Hyperthreading processors like the Xeon and the new Intel Pentium 4 make it possible to run two applications at the same time without any context switches and without the need to change the communication software. However, the processes compete for CPU-internal and network resources in addition to memory and I/O. Thus, interesting new options for time-shared execution are available but have to be handled carefully. The target architecture considered is a cluster with high-performance network (like Myrinet or Quadrics [35]) and user-level communication. In this paper, we only consider single-CPU nodes (hyperthreaded or standard) but our approach would be extendible to multi-way nodes. Considering the possibilities of hyperthreaded CPUs, we limit our coscheduling to a maximum of two jobs, i.e. a multiprogramming level of 2. In the following, we use the term coscheduling in the sense of running multiple jobs together.

The objectives for our own LOMARC job scheduler are:

- Inclusion of all relevant resources (CPU, network, disk, memory)
- Support of standard time-shared execution on standard CPU and coscheduling on hyperthreaded CPUs
- Increase of utilization though keeping basic primary goal of improved response times
- Usage of application characteristics via a-priori knowledge
- Usage of otherwise standard state-of-the-art scheduling approaches (priorities, backfilling etc.)

We address our objectives by the following innovative solutions:

- Optimizing the schedule by matching two applications whenever possible to share resources for high utilization
- Calculating estimates for response-time impact and utilization improvement while considering reordering to match jobs
- Including application characteristics (CPU, network, disk, memory) via an integrated cost model to estimate matchability and slowdowns
- Relaxing the scheduling order and sorting jobs more flexibly by permitting jobs to move ahead in the schedule if they pair well with other jobs though potentially to some extent pushing other jobs backward in the queue

We apply the standard per-job approach for scheduling jobs, i.e. do not attempt any global optimization. The reasons are that global optimization has a high – $O(n^3)$ – time complexity and that its benefit is even questionable, considering that the submissions are dynamic and, in standard approaches with priorities, the overall context of jobs changes permanently.

We have tested our approach via an event-based simulation and the workload described in [12], comparing it to standard space-shared job scheduling. Our tests include investigations of different heuristics, focusing either on utilization or response-time impact. We present a maximum slowdown model for the cases of resource competition and validate our estimates by practical tests with synthetic programs on a cluster with hyperthreaded CPUs.