Optimal Scheduling for Real-time Parallel Computer Systems

Leyuan Shi

Department of Industrial Engineering
University of Wisconsin-Madison
Madison, WI 53706

Phillip Q. Hwang

Office of the Chief Scientist
National Imagery and Mapping Agency
Bethesda, MD 20816-5003

Abstract

Abstract: We will describe our ongoing work in the field of optimal scheduling for real-time systems. We are primarily concerned with optimal task allocation and job scheduling for parallel computer systems. Many real-time task allocation and job scheduling problems are proven to be NP-hard. Recently, we proposed a randomized optimization framework for efficiently solving such NP-hard problems. The proposed method, the Nested Partitions (NP) method, has been proved to converge to global optimal solutions and it is also highly matched to emerging massively parallel processing capabilities.

1 Introduction

Parallel processing systems, consisting of numerous tightly- or loosely-coupled processors, represent an increasingly important class of high-performance computing environments that makes it possible to solve large and complex problems otherwise intractable. Fundamental to realizing these performance benefits is the design of scheduling policies which allocate processors among the parallel jobs submitted for execution in a manner that optimizes a selected objective function.

In discussing scheduling on parallel computers, there is an important distinction that we need to make. The scheduling problem discussed here is really a combination of two problems. First there is the scheduling of jobs on a set of processors. That is a set of processors is assigned to each job. In this context a job is an independent process to be performed. Each job may consist of several tasks or threads that can be performed sequentially or in parallel. Each of these tasks may have to communicate with any of the other tasks of that job, but not with other jobs. This defines the second problem, to schedule the tasks on the processors allocated to that particular job.

Usually, the task assignment problem is considered as deterministic optimization problem, since we often assume precise information about each task starting time, execution time, and the precedence relationship is known in advance. Thus, task assignment problem can be formulated as a deterministic optimization problem with the following format:

\[ \min_{\theta \in \Theta} F(\theta) \]  \hspace{1cm} (1)

where \( F(\theta) \) is the objective function of interest and \( \Theta \) is the constraint set.

For example, for \( P \) processors and \( M \) tasks, \( F(\theta) \) could be total error [8] or total tardiness [1] subject to the precedence, exclusion and deadline constraints.

However, for the job scheduling problem we often do not have the assumption of knowing everything in advance. On the contrary, we are usually interested in the situation where jobs arrive, either according to some distribution or in real-time. In either case we don’t know in advance when the next job is going to arrive. Therefore the job scheduling is considered as a stochastic optimization problem and often is more difficult than a deterministic case. The problem can be formulated as

\[ \min_{\theta \in \Theta} J(\theta) = E[L(\theta, \omega)] \]  \hspace{1cm} (2)

where \( J(\theta) \) is the average performance measure of interest, \( \Theta \) is the constraint set, \( L(\theta, \omega) \) is called the sample performance and \( \omega \) represents the stochastic effects of the system.

We have to realize that solving (2) is much more difficult than solving (1). That is because in most cases, there is no analytical expressions relating to the objective function \( J(\theta) \). Often, one has to resort to simulation to evaluate the objective function or to use real-time observations.

Most task assignment problems and job scheduling problems can be shown to be NP-hard [3, 7], and hence even for the deterministic problem, they must usually be solved using some heuristic method, such as genetic algorithms [5] or tabu search [4].

The last decade has witnessed a tremendous growth in the area of randomized (or probabilistic) algorithms. Randomized algorithms went from being a tool in computational number theory to finding widespread applications in many types of algorithms. Two benefits of
randomization have spearheaded this growth: simplicity and speed. Randomization avoids the use of complicated data structures, and yields efficient algorithms, both from the point of view of theoretical complexity as well as real-life usage. For example, computing the volume of a (high-dimensional) convex body is an ancient, basic, but extremely difficult task (going back to Archimedes). There are various negative results showing its difficulty. A breakthrough in the opposite direction was due to [2], in which the authors designed a polynomial time randomized algorithm to approximate the volume of a convex body in $n$ dimensions.

Motivated by the above observations and the new High-Performance Computing and Communications (HPCC) technologies, recently we have developed a new randomized optimization framework [9], the Nested Partitions Method which highly utilizes the parallel processing architectures. Through partitioning and random sampling, we are able to construct a Markov chain which has global optima as its absorbing states. Therefore, the algorithm will with probability one find a global optimum.

We believe that the proposed framework provides a fundamentally different setting to investigating randomized algorithms for solving an NP-hard optimization problem. Preliminary results show that the method is very efficient and robust. In this paper we aim at addressing both of these scheduling optimization problems in a unified framework.

2 The Nested Partitions Method

The proposed NP method can be briefly described as the follows. In each iteration of the algorithm we assume that we have a region (subset) of $\Theta$ that is considered the most promising. We then partition this most promising region into $M$ subregions, and aggregate the entire surrounding region into one region. At each iteration, we will therefore look at $M+1$ disjoint subsets of the feasible region $\Theta$. Each of these $M+1$ regions is sampled using some random (uniform) sampling scheme, and for each region a promising index is calculated. These promising indices are then used to determine which region is the most promising in the next iteration. If one of the subregions is found to be best, this region becomes the most promising region. If the surrounding region is found to be best, the region of one less depth than the current region becomes the most promising region. This new most promising one is then partitioned and sampled in a similar fashion.

In the first iteration we use the entire feasible region $\Theta$ as the most promising region. Since the surrounding region is empty, we only sample from $M$ regions in the first iteration, or in any iteration where $\Theta$ is considered the most promising region. It is also clear that since $\Theta$ is discrete, sooner or later we will have regions that contain only a single point in $\Theta$. We will call such regions, regions of maximum depth, and more generally, talk about the depth of any region. This is defined iteratively in the obvious manner, with $\Theta$ having depth 0 and so forth.

For example, consider a feasible region which consists of 8 points $\sigma_0 = \Theta = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and that in each iteration we partition the current most promising region into two disjoint sets (see Figure 1). At the first iteration, $\sigma_1 = \{1, 2, 3, 4\}$ and $\sigma_2 = \{5, 6, 7, 8\}$ are sampled. Assume that the promising index (the sampling result) in $\sigma_1$ is better than in $\sigma_2$. We then select $\sigma_1$ as the most promising region and further partition $\sigma_1$ to obtain $\sigma_3 = \{1, 2\}$ and $\sigma_4 = \{3, 4\}$. At the second iteration, $\sigma_3, \sigma_4$, and their surrounding region, $\sigma_2$ are sampled. If the promising index of $\sigma_3$ (or $\sigma_4$) is the best, then we select $\sigma_3$ to be the most promising region and partition $\sigma_3$ further into another two subregions $\sigma_5 = \{1\}$ and $\sigma_6 = \{2\}$ (or select $\sigma_4$ to be the most promising region and partition $\sigma_4$ into another two subregions $\sigma_7 = \{3\}$ and $\sigma_8 = \{4\}$). If the promising index of $\sigma_2$ is the best, then we select $\sigma_0$ as the most promising region. Now assume that $\sigma_3$ is the most promising region, at the third iteration, $\sigma_5, \sigma_6$, and their surrounding region $(\sigma_0 \setminus (\sigma_5 \cup \sigma_6))$ are sampled. If the promising index of $\sigma_5$ (or $\sigma_6$) is the best, then we select $\sigma_5$ (or $\sigma_6$) as the most promising region. If the promising index of the surrounding region is the best, we then select $\sigma_1$ as the most promising region. As the algorithm evolves, a sequence of the most promising region $\sigma(k)$ will be generated. Here $\sigma(k)$ is the most promising region in the $k$-th iteration. In [9], we have shown that $\sigma(k)$ is a Markov chain with all the global optima as its absorbing states. Thus, the global convergence will be guaranteed.

It needs to be pointed out that there are many variations of the NP method. For example, instead of retreat only to a region of one less depth, we can always make the entire feasible region (or any pre-specified region) the most promising region if the promising indices indicate a retreat is the appropriate move. This makes different transition probabilities for the algorithm to move from one region to another.

As we can see, the basic idea of the algorithm is to shift the focus from the feasible region $\Theta$, to a sequence of subsets of $\Theta$ and consequently it will lead to a global optimum. It is also clear from Figure 1 why the algorithm is termed a Nested Partitions algorithm.

This algorithm has several very attractive features: i) It does not require an initial solution, but takes into account the whole feasible region at the very beginning. This completely eliminates any initialization or guessing procedures for finding initial solutions. ii) It is also clear that the NP algorithm can take maximum advantage of the parallel computer architectures. In every iteration,
the algorithm looks at \( M + 1 \) regions, each of which can be handled independently in parallel. iii) The NP algorithm always spend the most computational effort in the region that is considered the most promising at any given iteration. Therefore, the NP algorithm will favor regions that have many good solutions and no bad solutions, rather than a region that has one very good solution and many bad ones. iv) The algorithm is also very generic and robust and can be applied to a wide variety of optimization problems, and as we have seen it is easy to implement and highly compatible to other optimization algorithms such as branch-and-bound and divide-and-conquer algorithms. v) It is also very flexible for limited computational budget, since we can always stop at a “good” region at any time. Or if we have the necessary resources, we can continue to seek out an optimal solution. This is a very attractive feature in practice.

Preliminary numerical results [9, 10] show that the NP algorithm is very efficient when it was applied to either deterministic optimization problem (1) or stochastic optimization problem (2).

3 Optimal Job Scheduling

As we can see that the NP method provides a generic framework for solving deterministic optimization problem (1) and stochastic optimization problem (2). In this section, we present an example to illustrate that how the NP method can be applied to job scheduling problems.

The first part of our work consists of formulating the job scheduling problem in such a way that it is appropriate for optimization. In practice it is often equally difficult to identify what to optimize as to perform the actual optimization. Consequently, we are developing a new hybrid equipartitioning policy (HEQ) for job scheduling, that lends itself naturally to optimization.

To illustrate our ideas we give a simple example of a hybrid equipartitioning policy, that contains as special cases both the quasi-dynamic policy described above, and the static equipartitioning policy. We assume the system has \( P \) identical processors, with a minimum of \( \theta_1 \) processors allocated to each job. The maximum number of jobs processed at a time is therefore \( N = \frac{P}{\theta_1} \). The optimal value of \( \theta_1 \) depends on the type of jobs being submitted to the scheduler. If the current jobs are massively parallel, a large value of \( \theta_1 \) is likely to be optimal. On the other hand, if most of the jobs are sequential, \( \theta_1 \) may be optimal to have \( \theta_1 \) equal to one. Jobs arrive continuously according to some arrival process that does not have to be determined. When a job arrives to the systems, the processors are either rescheduled to accommodate this job or it is placed on a single FIFO queue with capacity of \( K \) jobs. We will denote the number of jobs in the queue as \( m \), and the number of jobs being processed as \( n \). We will use the following rule to determine if a rescheduling occurs upon arrival of a new job: If the new arrival finds that

\[
m(N - n) > \theta_2
\]

the processors are rescheduled to include this new job. Otherwise the job is placed on the queue. Under this policy, rescheduling is more likely if there are many jobs waiting (i.e. the system is heavily loaded) and when the processors are poorly utilized by the current system load. The condition (3) is never satisfied if the maximum number of jobs are being processed \( (n = N) \) so rescheduling is never performed in this case. Rescheduling is also never performed if \( \theta_2 = KN \). Clearly the design space is \( \Theta = \Theta_1 \times \Theta_2 = \{1, 2, \ldots, P\} \times \{0, 1, \ldots, KN\} \) with \( \theta_2 = 0 \) corresponding to the quasi-dynamic policy described above, and \( \theta_2 = KN \) corresponding to the static policy. Notice that the size of \( \Theta_2 \) depends on the value of \( \theta_1 \) since \( N = \frac{P}{\theta_1} \). This policy contains both the static and dynamic policies as special cases, but also many in between policies. We can also implement the hybrid policy on-line to adopt to changing conditions that call for either the dynamic policy, the static policy or some other intermediate policy.

There have been many different scheduling policies proposed for the job scheduling problem. Our work focuses on space slicing policies.

Yet another example where real-time optimization of scheduling policies is desirable is given by Lehoczky et al. [6]. They consider the hard scheduling of aperiodic jobs along with periodic jobs. Several alternatives are proposed to deal with the aperiodic jobs, such as allowing
aperiodic jobs to interrupt periodic jobs, give aperiodic jobs their own resources, or giving aperiodic jobs lower priority to periodic jobs. Clearly none of these solutions is ideal. We plan to develop a hybrid policy incorporating all three of the above policies and use the NP method to find the optimal solution.

4 Conclusions

The NP method can be applied to both deterministic problems such as the task assignment problem and to stochastic problems such as optimizing the HEQ job scheduling policy in real-time. This new optimization method will therefore offer a unified framework for optimizing both aspects of the scheduling problem. The current status of this work is the following. We have formulated a generic version of the NP method for both deterministic and stochastic problems, and proved the global convergence of the NP method. Preliminary numerical results indicate that the method could be very efficient when applied to the scheduling problem above. We are currently working on establishing the convergence rate of the method on to specific problem structures. In particular we are looking at the two above mentioned scheduling problems. Although the asymptotic convergence results already obtained for the generic NP algorithm are very reassuring, we expect to get much stronger non-asymptotic results when the algorithm is formulated for a specific problem structure.

In conclusion, our work in real-time optimization is twofold. First there is the formulation of new scheduling policies. Guiding this part of our work is the goal of the policy being easily optimized in real-time so that it can adapt to changes in the workload and work characteristics. This strongly ties in with the second part of our work, which is to develop a new framework for optimization that is capable of optimizing both deterministic and stochastic problems. Guiding this part is the dual nature of scheduling on parallel computer systems, which demands method that are capable of optimizing both the deterministic task assignment problem and the stochastic jobs scheduling problem.

Acknowledgements

This work was partially supported by Naval Surface Warfare Center contract N00178-96-M3852.

References


