Deadline Scheduling for Animation Rendering

Eric Anderson, Dirk Beyer, Kamalika Chaudhuri, Terence Kelly, Norman Salazar,
Cipriano Santos, Ram Swaminathan, Robert Tarjan, Janet Wiener, Yunhong Zhou

Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto CA 94304
firstname.lastname@hp.com

Categories and Subject Descriptors: F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*Sequencing and Scheduling*

General Terms: Algorithms, Performance

Keywords: deadline scheduling, animation rendering

1. INTRODUCTION

We describe a new class of scheduling problem with precedence constraints, the *disconnected staged scheduling problem* (DSSP). DSSP is a nonpreemptive multiprocessor deadline scheduling problem; we seek to maximize the aggregate value of jobs that complete by a specified deadline. It arises in many commercially-important domains including bioinformatics and seismic signal processing.

Our interest in DSSP began with the practical problem of scheduling computer animation rendering jobs. Each job represents a brief film clip and consists of several stages that must be processed in order (e.g., physical simulation, model baking, frame rendering, and clip assembly). Each stage in turn consists of computational tasks that may be run in parallel; all tasks in a stage must finish before any task in the next stage can start. A job completes if and only if all of its tasks complete. Precedence constraints exist among tasks within a job, but not among tasks in different jobs. Jobs run overnight and yield value only if they complete before the artists who submitted them return the following morning. Demand frequently exceeds available CPU capacity, making it impossible to complete all submitted jobs by the deadline. The set of jobs is known in advance but their computational demands (e.g., the run times of tasks) are not precisely known.

Existing scheduling practices rely on priority schedulers, which are not well suited to DSSP because ordinal priorities cannot adequately express the value of jobs. Furthermore priority schedulers make *job selection* decisions as by-products of *task sequencing* decisions. Our approach is to assign to jobs *completion rewards* whose sums and ratios are meaningful, and to perform job selection and task sequencing separately.

We present both theoretical analysis and empirical evaluation of our DSSP solution. Our empirical results are based on an eight-week trace of 2,388 jobs collected in a 1,000-CPU production system that rendered part of film *Shrek 2* in 2004. We show that our two-phase method improves aggregate reward and achieves near-optimal performance under some conditions. A prototype of our scheduler is currently deployed at DreamWorks, one of Hollywood's top three animation studios. A full version of this paper will be available as an HP Labs technical report.

2. PROBLEM STATEMENT

Formally, DSSP consists of J jobs, indexed $j \in 1 \ldots J$. Job j contains G_j stages, indexed $g \in 1 \ldots G_j$. The set of tasks in stage g of job j is denoted S_{gj}. Stages encode precedence constraints within a job: no task in stage $g+1$ may begin until all tasks in stage g have completed. No precedence constraints exist among tasks in different jobs, i.e., the directed acyclic graph (DAG) of task precedence constraints is *disconnected*, with one component per job.

The execution time (or "length") of task i is denoted L_i. The total processing demand of job j, denoted $T_1(j)$, equals $\sum_{g=1}^{G_j} \sum_{i \in S_{gj}} L_i$. The *critical path length* of a job is denoted $T_\infty(j) \equiv \sum_{g=1}^{G_j} \max_{i \in S_{gj}} \{L_i\}$. Figure 1 illustrates two jobs.

At most one task may occupy a processor at a time, and tasks may not be preempted, stopped/re-started, or migrated after they begin. Let C_j denote the completion time of job j in a schedule. Let R_j denote its completion reward. Our goal is to sequence tasks onto processors to maximize aggregate reward $R_\Sigma \equiv \sum_{j=1}^{J} U_D(C_j)$, where $U_D(C_j) = R_j$ if $C_j \leq D$ and $U_D(C_j) = 0$ otherwise. This objective function is sometimes called "weighted unit penalty" [1].

The computational complexity of DSSP is formidable, even in a severely restricted special case. General DSSP is not merely NP-hard but also NP-hard to approximate within any polynomial factor, assuming that P $\neq$ NP. Furthermore, even the special case of DSSP with unit rewards and unit execution times is *strongly NP-complete.*

THEOREM 1. *General* DSSP *is NP-hard to approximate within any polynomial factor.*

THEOREM 2. *Unweighted* DSSP *with unit task execution time is strongly NP-complete.*

3. SOLUTION METHODS

Our approach decomposes DSSP into two tractable phases, an offline job selection phase followed by an online task sequencing phase.

The goal of job selection is to choose a subset of jobs with maximal aggregate completion reward subject to a processing capacity constraint. Binary decision variable $x_j = 1$ if job j is selected, $x_j = 0$ otherwise. P denotes the number of processors. Selection solves the following integer program:

$$\text{Maximize} \quad \sum_{j=1}^{J} x_j R_j \qquad (1)$$

$$\text{subject to} \quad \sum_{j=1}^{J} x_j T_1(j) \leq r \cdot PD \qquad (2)$$

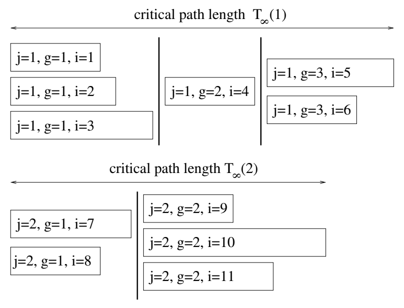

Figure 1: Job (j), task (i), and stage (g) structure for two jobs.

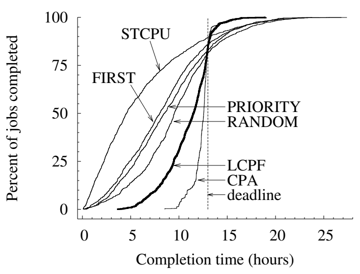

Figure 2: Distributions of job completion times.

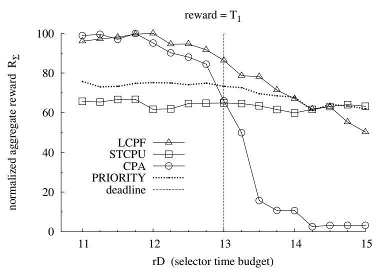

Figure 3: Selection plus dispatching performance.

PD in Equation 2 is the total amount of processor time available. Parameter r allows us to select a set of jobs whose total processor demand differs from the total available. Our selection problem is a classic 0-1 knapsack problem, for which a wide range of solvers exist [2]. We implemented three: the simple classic greedy heuristic, dynamic programming (DP) by profits, and a mixed integer programming (MIP) solver that can handle side constraints not discussed in this paper.

A dispatcher sequences tasks from selected jobs onto processors. We employ a non-delay (or "work-conserving") dispatcher that places a runnable job onto an idle processor whenever one of each is available. If there are several runnable tasks, a *dispatcher policy* chooses one. We empirically evaluated over two dozen dispatcher policies and present results for the best performers: RANDOM (choose randomly), FIRST (choose runnable task with highest ID), PRIORITY (choose highest-priority task), STCPU (choose a task from the *job* with the lowest total processing demand $T_1(j)$), and CPA (the critical path algorithm, sometimes called HLFET [3]). Our new dispatcher policy LCPF exploits the disconnected precedence DAG of DSSP; it chooses a task from the job with the longest critical path $T_\infty(j)$.

4. ANALYSIS

We prove worst-case performance bounds for two-phase approaches to unweighed DSSP (all jobs have unit completion reward). Our bounds depend on the maximum critical path length among all jobs, denoted $T_\infty^{\max}$.

MAXK is an offline algorithm that computes the maximum value K such that the K jobs with the highest $R : T_1$ ratio can be completed by the deadline. Given a value K, MAXK simply simulates dispatching all tasks of the selected jobs to check whether all of them complete by the deadline; linear or binary search can be used. We now prove that MAXK computes near-optimal schedules for unweighted DSSP.

THEOREM 3. *Algorithm* MAXK *can schedule at least* $\max\left\{\left(1-\frac{T_\infty^{\max}}{D}\left(1-\frac{1}{P}\right)\right) OPT-1, OPT-(P-1)\right\}$ *jobs.*

MAXK requires the execution times of each task, which are not always available. Our next result shows that two-phase solutions that do *not* require task lengths guarantee good results in the unweighted case if $T_\infty^{\max}/D$ is small.

THEOREM 4. *The two-phase solution using a greedy knapsack selector with the selection parameter* $r = 1-(1-1/P)(T_\infty^{\max}/D)$ *and any non-delay dispatcher completes at least* $(1-\frac{T_\infty^{\max}}{D}(1-\frac{1}{P}))OPT-1$ *jobs before the deadline.*

Theorem 4 implies that any two-phase solution (with a proper selection parameter r) completes at least half as many jobs as an optimal algorithm if $T_\infty^{\max} \leq D/2$. As $T_\infty^{\max}/D$ goes to 0, its performance approaches optimal.

5. EXPERIMENTAL RESULTS

The bound of Theorem 4 is weak when $T_\infty^{\max}/D$ is large. In this section we show experimentally that our two-phase approach yields good schedules for trace inputs with relatively high $T_\infty^{\max}$, and that our method outperforms existing practices substantially. Our experiments use two parameters from the DreamWorks cluster where our traces were collected: $P = 1,000$ processors and deadline $D = 13$ hours.

We first compare dispatcher performance alone. Figure 2 shows the distribution of job completion times for six dispatcher policies. LCPF and CPA overtake the other policies shortly after the deadline. However, by reducing selection parameter r we can shift their curves to the left, i.e., we can control when LCPF overtakes the other policies. For our animation workload, $r \approx 0.9$ yields good results. LCPF and CPA perform much better than the others in terms of makespan (time to finish last job). LCPF is preferable because it completes far more jobs early in the evening than CPA.

We also evaluated a complete two-phase scheduler (selector plus dispatcher) in terms of aggregate reward R_Σ. These tests used traces from the ten nights whose submitted jobs had the greatest total processor demand $\sum T_1(j)$. We used our MIP to select jobs, varying selection parameter r to see how under- or over-selection impacts aggregate rewards.

Figure 3 presents results for one set of experiments; others are described in the full paper. The horizontal axis shows the time budget $r \cdot D$ used during selection. Our results show that LCPF outperforms the other policies if r is tuned. CPA is comparable to LCPF if r is well tuned, but it suffers far more than LCPF when r is poorly tuned. STCPU and PRIORITY are relatively insensitive to r but yield considerably lower aggregate reward than well-tuned LCPF. In terms of aggregate reward, LCPF outperforms PRIORITY by 9%–32% in our experiments.

6. REFERENCES

[1] P. Brucker. *Scheduling Algorithms.* Springer, 3rd edition, 2001.

[2] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer, 2004.

[3] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, Dec. 1999.