RC 20104 (06/19/95)
Computer Science / Mathematics

IBM Research Report

Preemptive Scheduling of Parallel Tasks

Uwe Schwiegelshohn

University of Dortmund
Dortmund, Germany

John Turek and Joel L. Wolf

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

Abstract:

Consider a system of independent parallel tasks which are to be scheduled on a parallel computer. Preemptive execution of the tasks is allowed as long as there is no task migration between the processors, and all subtasks associated with a task are active at the same time. For each task the number of processors, the execution time and a weight are known. The problem is to find a schedule that minimizes the mean weighted flow time. Recently, a polynomial algorithm has been presented for this NP-hard problem which does not allow preemption and produces solutions within a fixed multiplicative constant (10.45) of the optimal non-preemptive schedule. It has also been shown that when all the tasks require only a single processor preemption does not reduce the cost of the optimal schedule. In this paper we show that when tasks run across multiple processors preemption can produce better solutions. We then describe a new algorithm with a bound of approximately 2.42 and show that this bound is tight for the algorithm. An interesting feature of our method is that it requires that at most two tasks be resident at any time on a given processor.

1 Introduction

Consider a parallel computer composed of P processors, on which a system τ consisting of M independent tasks is to be scheduled. Assume that for any task $i \in \{1, ..., M\}$ we are given the number of processors $w_i \leq P$ required by the task, the running time $h_i > 0$ of the task and a task weight u_i. All of the processors allotted to a task are required to execute that task in unison. However, preemption is allowed as long as all processors acting on the task perform the task switch at the same time. That is, task i must always be started, interrupted, resumed and completed by all w_i processors at the same time, respectively. As a result, associated with each task i is a monotonic increasing sequence of time instances $t_i(0), t_i(1), \ldots t_i(2d_i - 1)$ where task i is started at time $t_i(0)$ and completed at the *completion time* $t_i(2d_i - 1)$. Note that $d_i - 1$ denotes the number of times that a task is interrupted and resumed. We will use the convention that task i is interrupted at the time instances denoted by $t_i(2\delta - 1)$ and resumed at the time instances denoted by $t_i(2\delta)$ for $0 < \delta < d_i$. Because tasks are not allowed to migrate, we assume that this kind of preemption does not incur any time penalty and therefore $\sum_{\delta=1}^{d} t_i(2\delta - 1) - t_i(2\delta - 2) = h_i$.[1] A *schedule* consists of an even sequence of time instances $t_i(\mu)|0 \leq \mu < 2d_i$ and a set $\mathcal{P}_i \subseteq \{1, 2, \ldots P\}$ for each task i such that $t_i(\mu) < t_i(\nu)$ if $\mu < \nu$, $\sum_{j=0}^{d_i-1} t_i(2j+1) - \sum_{j=0}^{d_i-1} t_i(2j) = h_i$, and $|\mathcal{P}_i| = w_i$. The schedule must be *legal* in the sense that for any time t no processor is used twice. In other words,

$$\mathcal{P}_j \cap \mathcal{P}_k = \emptyset \ \ \forall j, k \in \{i | t_i(2\delta) \leq t < t_i(2\delta + 1),\ and\ 0 \leq \delta < d_i\}\ and\ j \neq k$$

for all t.

Our goal is to find a schedule that minimizes the weighted average response time

$$\sum u_i t_i(2d_i - 1).$$

This problem is known to be NP-hard in the strong sense for arbitrary P even when the number of processors, w_i, required by each is 1 [1]. Thus, efforts to deal with this problem have so far been devoted to finding good approximation algorithms. However, most previous research has again concentrated on the classical problem dealing with nonparallel tasks that run on a single processor. Kawaguchi and Kyan [2] and the references therein provide a good overview of this area of study. We will briefly present some of their results in the next section. More recently, Turek et al. [3] presented an algorithm called SMART which provides an approximation algorithm for the case where the weights u_i are 1 while the widths can take arbitrary integer values. This algorithm does not allow task preemption and has subsequently been extended to address the general case [4]. In its revised form the algorithm has a time complexity of $O(M \log M)$ and an approximation factor of 10.45.

In this paper we devise a new algorithm with an approximation factor of 2.42 by allowing preemption of the tasks. Like the non-preemptive algorithm presented in [4] the running time of our algorithm is $O(M \log M)$.

In Section 2 we review some of the results developed in the area of scheduling non-parallel tasks. These results are used extensively in the rest of our paper. We also provide an example to show that a schedule with smaller cost can be obtained by

[1] One can informally think of the execution of task i as taking place within a vertically stacked sequence of rectangles whose width w_i stretches along a *processor* axis and whose combined heights sum up to h_i and stretch along a *time* axis.

allowing preemption. The algorithm is described in Section 3. In the next section we prove the approximation factor of 2.42 and show that this bound is tight. In Section 5 we compare the properties and requirements of our algorithm with the features of modern parallel computers. Section 6 contains conclusions and an outlook for further work.

2 Scheduling Nonparallel Tasks

Scheduling of nonparallel tasks to minimize average response time has received a fair amount of attention in the literature. Kawaguchi and Kyan [2] present an algorithm with a running time of $O(M \log M)$ for this special case of our problem. First, they generate a priority list of tasks by arranging the tasks in nonincreasing order of $\frac{u_i}{h_i}$. This ratio is also known as Smith's ratio [5]. Whenever a processor becomes available they assign the next task in this order to the free processor until all tasks are completed. This schedule is clearly a *list schedule* and they call it the LRF (*largest ratio first*) schedule. They show that LRF produces a schedule with an approximation factor on the average response time of $(\sqrt{2}+1)/2$. Moreover, they show that this bound is tight.

McNaughton [6] proved that, regardless of cost metric, the cost of the optimal schedule cannot be reduced by introducing preemption without task migration. Although this observation has significant consequences for the scheduling of nonparallel tasks, the observation does not hold for the general case we are studying as is demonstrated by the example shown in Figure 1.

Given are 3 tasks with $w_1 = P$, $h_1 = 3$, $w_2 = P - 1$, $h_2 = 2$, $w_3 = 1$ and $h_3 = 6$. All tasks have the weight 1. In an optimal non-preemptive schedule task 2 is started concurrently with task 3. This block and task 1 are then scheduled in any order, resulting in a cost of 17. In the optimal preemptive schedule however, first task 2 and task 3 are started together. Then at time $t = 2$ task 3 is interrupted and task 1 is executed before execution of task 3 is resumed at time $t = 5$. The cost of this schedule is only 16.

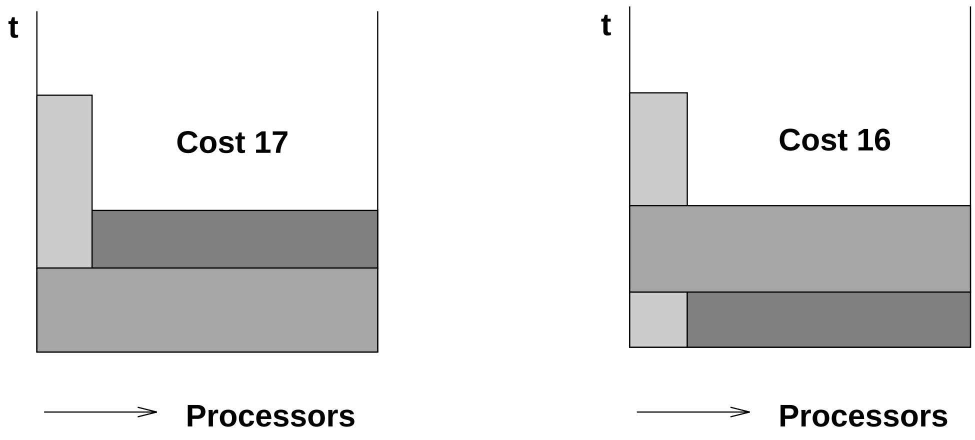

Figure 1: Non-preemptive versus Preemptive Schedules for Parallel Tasks

3 The Algorithm

In this section we present our preemptive algorithm, called *PBI*: *P*reemptive *b*y *I*nterleaving. Intuitively, we generate two parallel schedules S_1 and S_2 such that schedule S_2 will contain only a single task at any time. This single task has the property that it would not have fit on S_1 at the same time. Then both schedules are executed on one set of processors by interleaving them.

The details of the PBI algorithm are given below:

1. Generate a priority list by ordering all tasks by Smith's ratio such that task i precedes task j if $\frac{u_i}{w_i t_i} > \frac{u_j}{w_j t_j}$.

2. When less than P processors are used in both S_1 and S_2 combined, first move any active task on S_2 to S_1. Then schedule the next task i in the priority list on S_1 if at least w_i processors are available on S_1. Otherwise schedule i on S_2.

3. Divide S_1 and S_2 into horizontal stripes such that the end of each stripe is defined by either the completion of one task in S_2 or the movement of a task from S_2 to S_1 or the end of the last task in S_1.

4. Generate a combined schedule by alternatively executing the stripes of S_1 and S_2 in the given order starting with the first stripe of S_1.

Before addressing the approximation factor of PBI, some other properties are mentioned:

- S_1 is a non-preemptive schedule based on tasks or parts of tasks of τ.
- S_2 always contains a single task or part of a single task in each stripe except for the last stripe which might be empty. This task always stretches from the beginning to the end of the stripe. Therefore, the allocation of processors to tasks is unconstrained in S_2.
- No task can ever move from S_1 to S_2. Consequently, no two stripes in S_2 can contain the same task.

This guarantees that for any task which is executed on S_1 as well as on S_2 the same processor allocation can be used. Thus, the resulting preemptive schedule does not require any task migration.

Since PBI uses two way interleaving of two non-preemptive schedules, no more than two tasks are ever resident at the same time on one processor.

The algorithm is also described in Figure 2. Note that the context switch can be avoided in many cases by moving separate horizontal parts of tasks together. More precisely, using this optimization any task will be executed as early as possible subject to the following three constraints:

1. The processor allocation of the PBI schedule remains invariant.
2. If $t_i(0) \geq t_j(2d_j - 1)$ holds for any two tasks i and j in the PBI schedule with $\mathcal{P}_i \cap \mathcal{P}_j \neq \emptyset$ this must also be true in the optimized schedule.
3. The completion time of no task can increase.

An example of this optimization is also given in Figure 2. It can be easily seen that only in case of tasks which complete in S_2 a context switch may be necessary. If therefore each task is executed at least partly in S_1 then the above mentioned optimization will result in a non-preemptive overall schedule. The cost of the schedule may also be reduced by the optimization. However the worst case is not affected, as is shown in the next section. For reasons of completeness we would like to mention two previously reported results in connection with priority lists:

- Any list schedule based on some ratio of height, width and weight will not guarantee a constant approximation bound for the non-preemptive case [3].
- It is possible to obtain an approximation bound of 2 by using a priority list with Smith's ratio and restricting the maximum width of any task to $\frac{P}{2}$ [7].

4 The Approximation Factor

We define s_τ, p_τ, and o_τ to be the costs of the schedule generated by PBI, an optimal preemptive schedule and an optimal non-preemptive schedule, respectively, for any task system τ.

Our main result in this section is the following:

Theorem 4.1 *Consider a task system $\tau = \{(h_i, w_i, u_i) | i = 1, ..., M\}$. The PBI algorithm has time complexity $O(M \log M)$ and satisfies $s_\tau / p_\tau \leq \sqrt{2} + 1$.*

Proof: The time complexity is determined by the generation of the priority list which directly yields the claim. Note that the actual processor allocation is not part of the algorithm. Its inclusion is straightforward and would result in a worst case time complexity of $O(\sum_{i=1}^{M} w_i)$.

For the proof of the approximation factor we consider a transformation of task system τ into a width 1 task system $\bar{\tau}$. This is achieved by dividing task i of τ into w_i identical tasks of width 1, height t_i and weight $\frac{u_i}{w_i}$. Note that Smith's ratio is the same for any task j of $\bar{\tau}$ and for its parent task i of τ. We can now use Kawaguchi and Kyan's LRF algorithm [2] to generate a schedule for $\bar{\tau}$ on P processors with costs $c_{\bar{\tau}} \leq \frac{\sqrt{2}+1}{2} o_{\bar{\tau}}$. See Figure 3.

Next, we note that the costs of the LRF algorithm for $\bar{\tau}$ are not smaller than the combined costs of the parallel schedule on S_1 and S_2. The latter are at least half the costs of the PBI schedule, since our approach of 2-way interleaving can at most double the execution time of any task.

Altogether, we can therefore conclude:

$$s_\tau \leq 2c_{\bar{\tau}}$$

$$
\begin{aligned}
&\leq (\sqrt{2}+1)o_{\bar{\tau}} \\
&= (\sqrt{2}+1)p_{\bar{\tau}} \\
&\leq (\sqrt{2}+1)p_{\tau}
\end{aligned}
$$

This completes the proof.

■

In the next lemma we address the question of tightness of the bound in Theorem 4.1.

Lemma 4.1 *There is a task system τ for which* $\lim_{P\to\infty} \frac{s_\tau}{p_\tau} = \sqrt{2}+1$.

Proof: We use a task system which is partly based on the width 1 task system proposed by Kawaguchi and Kyan to show the tightness of their bound. More precisely, we have five different kinds of tasks:

1. 1 task with $h_i = 1+\sqrt{2}-\frac{1}{n}$, $w_i = P$, and $u_i = \frac{1}{n}$.
2. 1 task with $h_i = 1+\sqrt{2}$, $w_i = \lfloor(\frac{2-\sqrt{2}}{2})P\rfloor$, and $u_i = h_i w_i$.
3. $n(P+1)$ tasks with $h_i = \frac{1}{n}$, $w_i = 1$, and $u_i = h_i$.
4. n tasks with $h_i = \frac{1}{n^2}$, $w_i = P$, and $u_i = h_i w_i$.
5. $n-1$ tasks with $h_i = \frac{1}{n^2}$, $w_i = 1$, and $u_i = h_i$.

Note that Smith's ratio is 1 for all tasks except the first one for which it is very small. By using small variations of the weight we make sure that in our priority list every $P+1$ tasks of group 3 are followed first by 1 task of group 4 and then by 1 task of group 5 as long as one is available. Both tasks of group 1 and 2 form the end of the list. The resulting schedule and the optimal schedule can be seen in Figure 4.

For $n, P \to \infty$ the cost of the PBI schedule is approximately $2(1+\sqrt{2})(2+\sqrt{2})\frac{2-\sqrt{2}}{2}P + P = (3+2\sqrt{2})P$.

In the optimal schedule all tasks of group 4 are scheduled first, followed by the group 2 task parallel with the tasks of group 3 and 5. The single task of group 1 is scheduled at the end. Also assuming $n, P \to \infty$ the cost of this schedule is approximately $(1+\sqrt{2})^2\,\frac{2-\sqrt{2}}{2}P + \frac{\sqrt{2}}{2}P = (1+\sqrt{2})P$.

The ratio between these costs is $1+\sqrt{2}$. ■

5 Use for Parallel Computers

Implementing preemptive scheduling on modern parallel machines may cause the following problems:

1. If tasks are executed on different sets of processors the costs of migrating the tasks from one processor set to another will increase the execution time of these tasks and potentially of other tasks due to network clogging problems. Therefore in this case, the assumption that executing a task in a preemptive fashion requires the same costs as non-preemptively executing this task is no longer true.

2. The number of tasks which are resident on a processor cannot be arbitrarily big if the processor is using its own local memory to keep the code and the data.

3. For massively parallel computers the communication delay between two processors is significant compared to the cycle time. Therefore, after interrupting a task the computer must be aware that messages for this task may still come in. This also makes task migration very difficult.

Our algorithm has addressed the first two items in the preceding list by not allowing task migration and by guaranteeing that no more than two tasks are resident on any processor at any instance of time. Regarding the last item it should be noted for instance that the Connection Machine CM-5 has the ability to perform a common context switch for all processors. Such a feature could be used to implement a preemptive algorithm like PBI, as here the context switch is also performed simultaneously for all processors. This is also true for the checkpoint restart facility which can be found in many multiprocessors.

Moreover in many parallel computers, there is a distinction between the compute agent and the communication agent in a processor node. For instance, the Intel Paragon has separate processors dedicated to each task. In such an environment the context switch would only be performed by the compute agent while the communication agent is still able to receive messages for the now waiting task. Of course this requires that both data sets are kept in memory and therefore restricts the number of tasks resident on a specific processor as well.

6 Conclusions:

In this paper we have:

1. Developed a new preemptive algorithm for the mean weighted flow-time problem with parallel tasks. This algorithm has an approximation factor of 2.42 which compares favorably to the approximation factor of 10.45 for the only other known (non-preemptive) algorithm to address this problem [4].

2. Given an example to show that the bound of 2.42 is tight for the algorithm.

We have further addressed the problem of implementing this kind of algorithm on parallel computers. In contrary to the SMART algorithm [4] the new algorithm is based on a priority list. Therefore, it is better suited for on–line scheduling. However, it should be noted that by extending this algorithm to on–line scheduling it is not possible to maintain the bound, assure that no task migration is required and to limit the number of tasks assigned to one processor at the same time.

References

[1] Garey, M. and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[2] Kawaguchi, T. and S. Kyan. Worst Case Bound of an LRF Schedule for the Mean Weighted Flow-Time Problem. *SIAM Journal on Computing*, 15(4):1119–1129, November 1986.

[3] Turek, J., U. Schwiegelshohn, J. Wolf, and P. Yu. Scheduling Parallel Tasks to Minimize Average Response Times. In *Proceedings of the SIAM Symposium on Discrete Algorithms, Arlington, VA*, pages 112–121, January 1994.

[4] Schwiegelshohn, U., W. Ludwig, J. Wolf, J. Turek, and P. Yu. Smart SMART Bounds for Weighted Response Time Scheduling. Technical Report RC 19789 (87176), IBM Research Division, July 1994.

[5] Smith, W. Various Optimizers for Single-Stage Production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[6] McNaughton R. Scheduling with Deadlines and Loss Functions. *Management Science*, 6(1):1–12, October 1959.

[7] Turek, J., W. Ludwig, J. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. Yu. Scheduling Parallizable Tasks to Minimize Average Response Time. In *Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, NJ*, pages 200–209, June 1994.

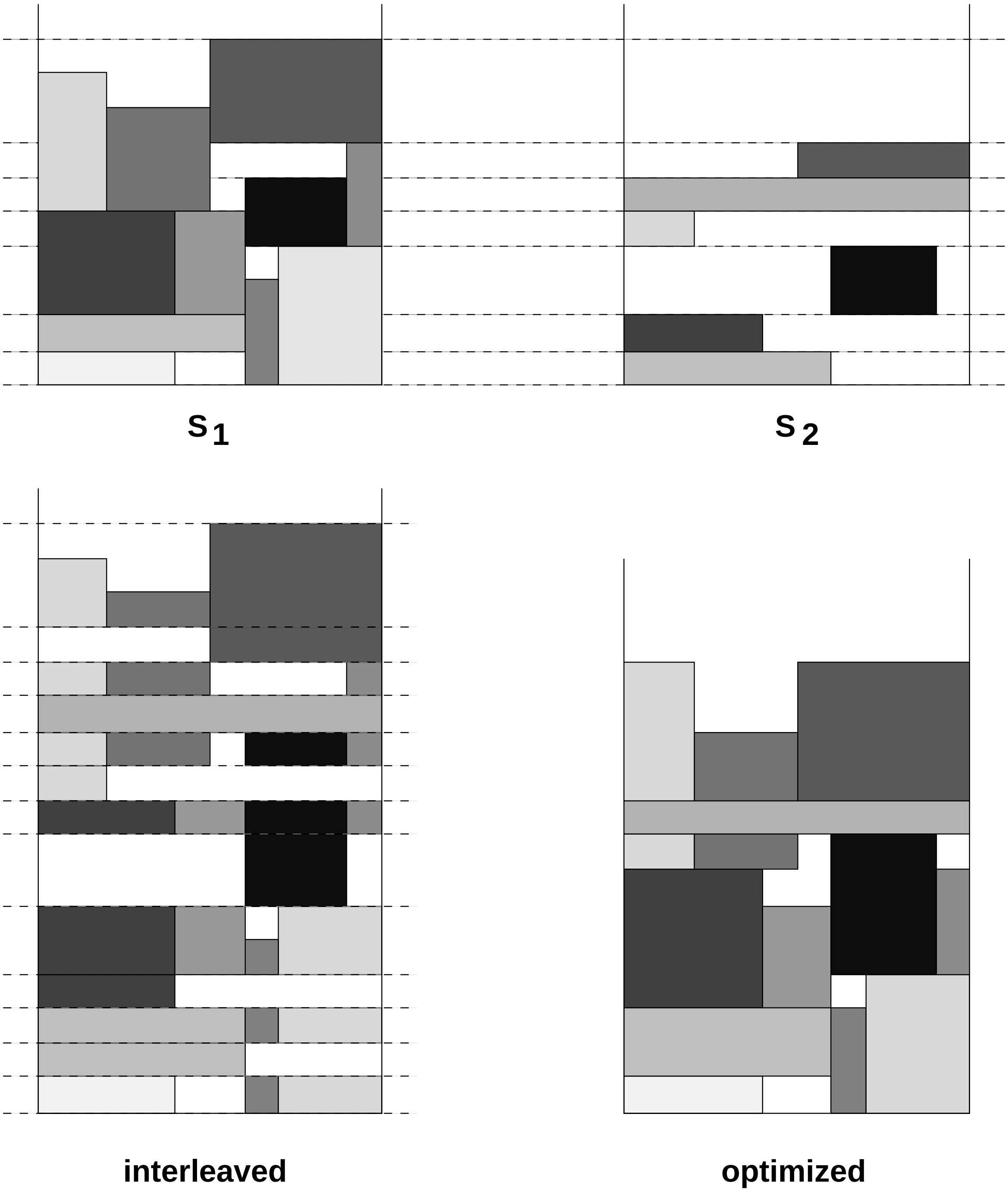

Figure 2: Schedules S_1, S_2, the Interleaved Schedule, and the Optimized Schedule

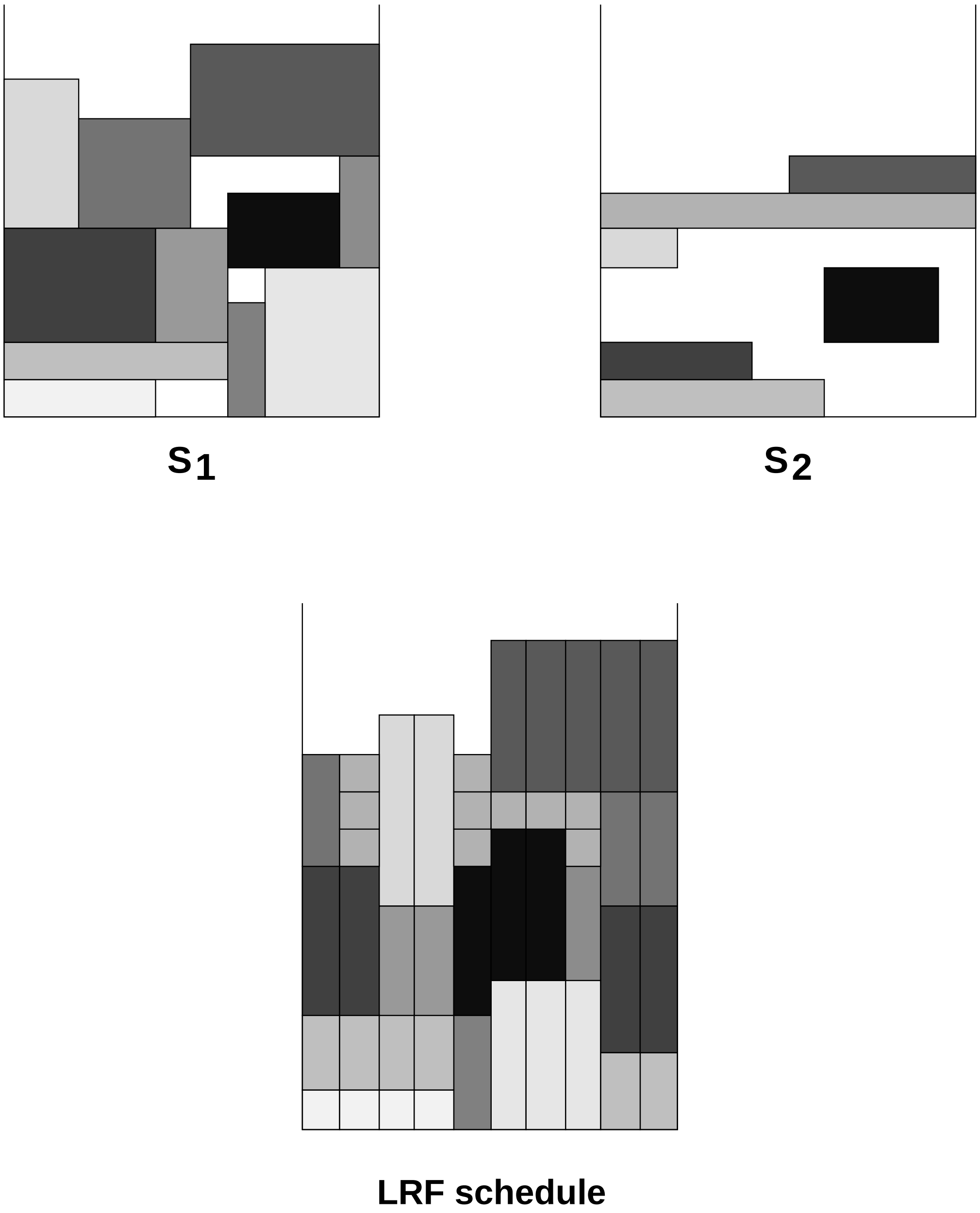

Figure 3: Schedules S_1 and S_2 and the LRF Schedule

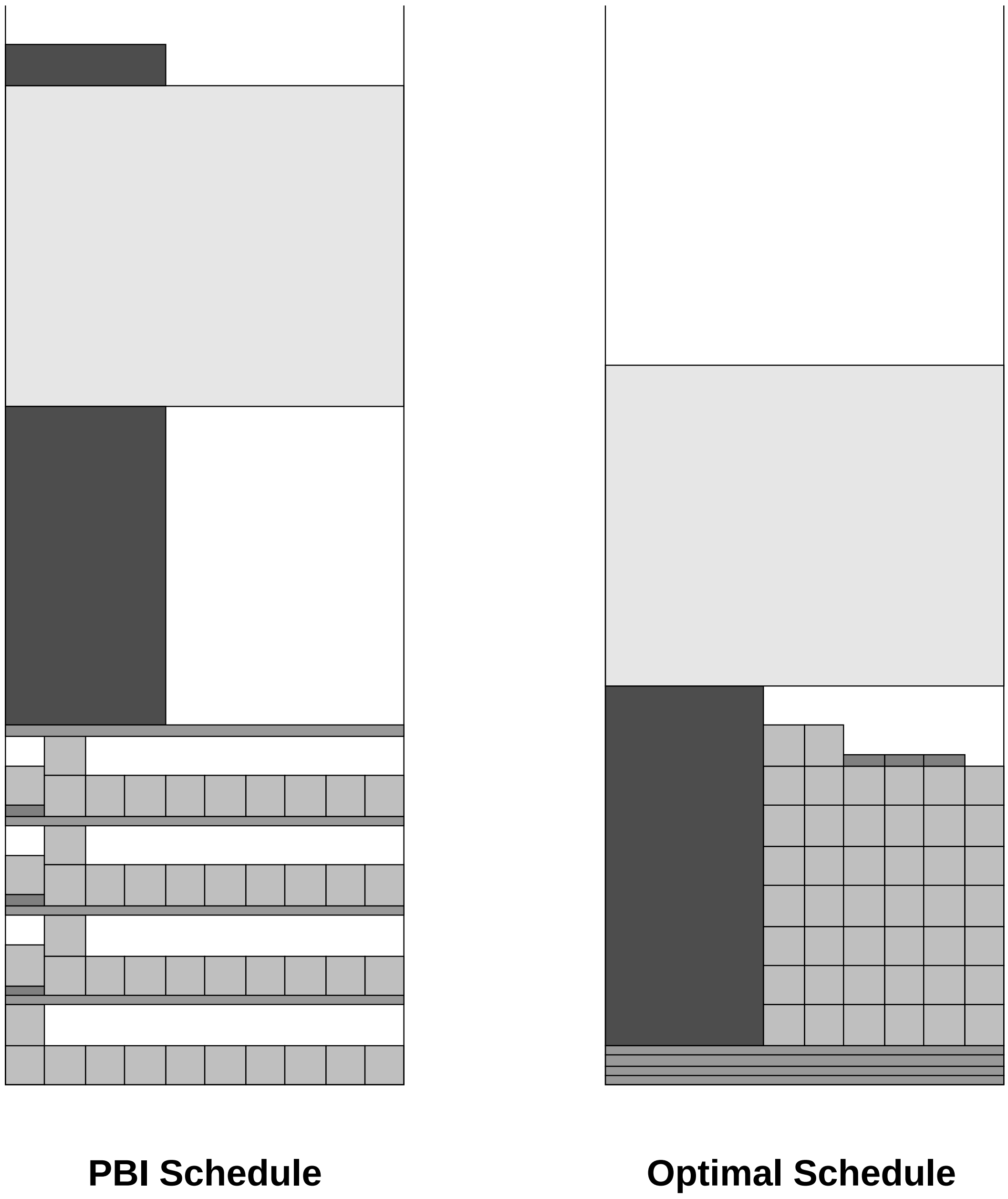

Figure 4: PBI Schedule and Optimal Schedule for Worst-Case Example