Analysis of First-Come-First-Serve Parallel Job Scheduling [*]

Uwe Schwiegelshohn[†] Ramin Yahyapour[‡]

Abstract

This paper analyzes job scheduling for parallel computers by using theoretical and experimental means. Based on existing architectures we first present a machine and a job model. Then, we propose a simple on-line algorithm employing job preemption without migration and derive theoretical bounds for the performance of the algorithm. The algorithm is experimentally evaluated with trace data from a large computing facility. These experiments show that the algorithm is highly sensitive on parameter selection and that substantial performance improvements over existing (non-preemptive) scheduling methods are possible.

1 Introduction

Todays massively parallel computers are built to execute a large number of different and independent jobs with a varying degree of parallelism. For reasons of efficiency most of these architectures allow *space sharing*, i.e. the concurrent execution of jobs with little parallelism on disjoint node sets. This produces a complex management task with the scheduling problem, i.e. the assignment of jobs to nodes and time slots, being a central part. Also, in a typical workload it can neither be assumed that all jobs have the same properties nor that there is a random distribution of jobs with different properties, see e.g. Feitelson and Nitzberg [3].

As most scheduling problems have been shown to be computationally hard, theoretical research has centered around proofs of NP-completeness, approximation algorithms, and heuristic methods to obtain optimal solutions. At the same time, the experimental evaluation of various scheduling heuristics, the study of workload characteristics, and consideration of architectural constraints have been the focus of applied research in this area.

But the interaction between both groups has been rather limited. For instance, so far very few algorithms from the theoretical community have been implemented within real schedulers. Similarly, heuristics used in real parallel systems have rarely been subject of a theoretical analysis. Various reasons are frequently cited to be responsible for the lack of interaction between both communities. For instance, designers of commercial schedulers do not care much about approximation factors. For them, deviations of factor 5 or factor 100 from an optimal solution will usually both be unacceptable for real workloads. Also, applied researchers often claim that the models and optimality criteria used in theoretical research rarely match the restrictions of many real life situations. On the other hand, as the worst case behavior of those heuristics typically found in commercial schedulers is usually quite bad with respect to the criteria often used in theoretical research, these algorithms are of little interest to theoreticians.

In our paper we want to demonstrate that there are algorithmic issues in job scheduling where theoretical and applied research can both contribute to a solution. To this end we discuss *First-come-first-serve* (FCFS) scheduling, a simple job scheduling method which can often be found in real systems but is usually considered inadequate by many researchers. First, we derive a job scheduling model based on the IBM RS/6000 SP2 as described by Hotovy [7] and address scheduling objectives. Then we show that for many workloads good performance can be expected from an FCFS schedule. Moreover, bad utilization of the parallel computer can be prevented by introducing gang scheduling into FCFS. We also demonstrate that the term fairness can be transformed into a specific selection of job weights. Using this weight selection we can prove the first constant competitive factor for general on-line weighted completion time scheduling where submission times and job execution times are unknown. Finally, we use simulation experiments with real workloads to determine good and bad strategies for FCFS scheduling with preemption. As the performance of the strategies is highly dependent on the workload, we conclude that an adaptive scheduling strategy may produce the best results in job scheduling for parallel computers.

2 The Model

Our model is based on the IBM RS/6000 SP parallel computer. We assume a massively parallel computer consisting of R identical nodes. Each node contains one or more processors, main memory, and local hard disks while there is no shared memory. Equal access from all nodes to mass storage is provided. Fast

[*]Supported by a grant from the NRW Metacomputing project

[†]Computer Engineering Institute, University Dortmund, 44221 Dortmund, Germany, uwe@ds.e-technik.uni-dortmund.de

[‡]Computer Engineering Institute, University Dortmund, 44221 Dortmund, Germany yahya@ds.e-technik.uni-dortmund.de

communication between the nodes is achieved via a special interconnection network. This network does not prioritize clustering of some subsets of nodes over others as in a hypercube or a mesh. Therefore, the parallel computer allows free variable partitioning [4], that is the resource requirement r_i of a job i can be fulfilled by each node subset of sufficient size. Further, the execution time h_i of job i does not depend on the assigned subset.

The parallel computer also supports gang scheduling [4] by switching simultaneously the context of all processors belonging to the same partition. This context switch is executed by use of the local processor memory and/or the local hard disk while the interconnection network is not affected except for message draining [16]. The context switch also causes a preemption penalty mainly due to processor synchronization, message draining, saving of job status and page faults. In the model we describe this preemption penalty by a constant time delay p. During this time delay no node of an affected partition is able to execute any part of a job. Note that the context switch does not include job migration, i.e. a change of the node subset assigned to a job during job execution.

Individual nodes are assigned to jobs in an exclusive fashion, i.e. at any time instant any node belongs at most to a single partition. This partition and the corresponding job are said to be active at this time instant. This property assures that the performance of a well balanced parallel job is not degraded by sharing one or a few nodes with another independent job. As gang scheduling is used, a job must therefore be either active on all or on none nodes of its partition at the same time.

As this simple model does not perfectly match the original architecture, we briefly discuss the deviations of our model from actually implemented IBM RS/6000 SP computers:

1. There are 3 different types of nodes in IBM SP2 architectures: *thin nodes*, *wide nodes*, and *high (SMP) nodes*. A wide node usually has some kind of server functionality and contains more memory than a thin node while there is little difference in processor performance. Recently introduced high nodes contain several processors and cannot form a partition with other nodes at the present time. However, most installations contain predominantly thin nodes.

2. Nodes need not necessarily be equipped with the same quantity of memory. But in most applications the majority of nodes have the same or a similar amount of memory. While e.g. the Cornell Theory Center (CTC) SP2 contains nodes with memory ranging from 128 MB to 2048 MB, more than 90% of the nodes have either 128 MB or 256 MB [7].

3. Access to mass storage usually requires the inclusion of an I/O node to the partition which is typically a wide node.

4. Interactive jobs running only on a single processor must not be exclusively assigned to a node. However, during operation a parallel computer is usually divided into a batch and an interactive partition. In our paper we focus on the batch partition as the management of the interactive partition is closely related to the management of workstations.

5. While most present SP2 installations do not allow preemption of parallel jobs, IBM has already produced and implemented a prototype for gang scheduling [16].

Altogether, our model comes reasonably close to real implementations. Finally note that parallel computers may contain several hundred nodes (430 for the batch partition of the CTC SP2).

Next, we describe the job model of the SP2. At first the user identifies whether he has got a batch or an interactive job. As there typically are separate interactive and batch partitions [7] we restrict ourselves to batch jobs. Besides requesting special hardware the user can also specify a minimum number of nodes required and a maximum number of nodes the program is able to use. The scheduler will then assign to the program a number of nodes within this range. During execution of the program neither the number of nodes nor the assigned subset of nodes will change. Using the terminology of Feitelson and Rudolph [5] we have therefore a *moldable* job model and *adaptive* partitioning. A user may submit his job to one of several batch queues at any time. Each queue is described by the maximum time (wall clock time) a job in this queue is allowed for execution. No other information about the execution time h_i of a job i is provided. When a job exceeds the time limit of the assigned batch queue, it is automatically canceled. For our study we use the same model with two exceptions:

1. No special requests are allowed.

2. The exact number of required processors is given for each job.

Both restrictions are addressed in Section 4. A similar model as the one described above has also been used by Feldmann et al. [6].

3 The Scheduling Policy

In general a parallel job schedule S determines for each job i its starting time and the subset of nodes $\mathcal{R}_i$ assigned to this job. The starting time of a job must be greater than its submission time s_i and node subsets of two jobs executing concurrently must be disjoint. The completion time t_i of a job depends on the starting time, the execution time h_i and the number and lengths of preemptions.

If the amount of requested nodes exceeds the nodes available on the parallel computer, a scheduling policy is used to settle those resource conflicts. This scheduling policy typically has the objectives to minimize the idle time on the nodes and to implement a priority strategy.

The first objective is closely related to the completion time of the last job of a given job set τ, the so called makespan $m_S = \max_{i\in\tau} t_i$. It is therefore not surprising that makespan scheduling has been frequently addressed in theoretical scheduling, e.g. Feldmann et al. [6] demonstrated that for their PRAM model (which is similar to the IBM model) a competitive factor of 2 can be guaranteed. As pointed out by Shmoys et al. [12] this method can also be modified to include arbitrary submission times. However, there is only little room for implementation of a priority strategy.

The second objective is not as clearly defined. It is based upon the intentions of the owner of a parallel computer. The owner may prioritize highly parallel jobs over mostly sequential jobs or the other way around (throughput maximization). Some users may receive a larger allocation of compute time while others are only allowed to use selected time slots. Altogether, it is possible to think of a large number of different strategies. In theoretical scheduling the criterion 'minimization of the sum of the weighted completion times' ($c_S = \sum_{i\in\tau} w_i t_i$) is frequently used. It is based on the observation that most users like to receive the results of their jobs as soon as possible (turnaround time minimization). By use of the weight w_i for each job i some jobs may indirectly get a higher priority than others. This criterion has recently gained attention for the scheduling of parallel jobs. In the off-line case ($s_i = 0$ for all jobs) Schwiegelshohn et al. [11] gave a deterministic algorithm with an approximation factor of 8.53. Chakrabarti et al. [1] derived a similar bound of 8.67 for a randomized method and an on-line problem (unknown submission times). Schwiegelshohn [10] was able to obtain an improved bound of 2.37 for the off-line problem by using preemption. Finally, Deng et al. [2] proved an approximation factor of 2 for their preemptive on-line model (unknown job execution times) with unit weights.

At this point we would like to address the difference between weighted completion and weighted flow time (= *completion time* − *submission time*). Obviously, the user is only interested in the time delay between job completion and job submission. However, it is much harder to approximate the optimal weighted flow time than the optimal weighted completion time, as shown by Leonardi and Raz [8]. Therefore, many theoretical researchers focused on the completion time problem. This may be justified as both measures differ from each other by only a constant. Therefore, it is sufficient to consider just one of both criteria for the purpose of comparing two schedules.

Further, it is emphasized by the previous paragraphs that addressing either the makespan or the weighted completion time criterion is not good enough. While an optimal makespan will not guarantee a minimal or even small sum of weighted completion times and vice versa [10], Stein and Wein [14] as well as Chakrabarti et al. [1] showed recently that there are methods which provide good performance for both criteria. Moreover, all the weighted completion time algorithms mentioned above fit in this category.

But even if the multiple criteria problem can be solved, the priority strategy must still be linked with the necessary selection of job weights. Hotovy [7] pointed out that in a real machine environment the selection of a priority strategy and its implementation is a complex and interactive process. Still, the overall goal of the policy established by the CTC is never explicitly stated and the final strategy is a result of several experiments which were evaluated with respect to backlog and average waiting (flow) time. A similar problem was also addressed by Lifka [9] when he described the properties of his *universal scheduler*.

Influenced by Lifka's work we use a priority policy with the simple objectives fairness and accounting for costs besides minimization of average completion (flow) time. We demand that the cost of a job i is only based on the amount of resources consumed. This way there is no gain in splitting a parallel job into many sequential jobs or in combining many independent short jobs into one long job or vice versa. Only the overhead of a parallel job in comparison to its sequential version must be paid for. Based on the results of Smith [13] this suggests the following weight definition:

DEFINITION 3.1. *The weight w_i of a job i is the product of the number of nodes r_i assigned to the job and the execution time h_i of the job.*

The weight in Definition 3.1 also represents the cost of a job. Note that this weight of a job is not available at its submission time if the execution time of the job is not known at this moment.

In many commercial schedulers fairness is further observed by using the *First-come first-serve* principle [7]. But starting jobs in this order will only guarantee fairness in a non-preemptive schedule. In a preemptive schedule a job may be interrupted by another job which has been submitted later. In the worst case this may result in job starvation, i.e. the delay of job completion for a long period of time. Therefore, we introduce the following parameterized definition of fairness:

DEFINITION 3.2. *A scheduling strategy is κ-fair if all jobs submitted after a job i cannot increase the flow time of i by more than a factor κ.*

It is therefore the goal to find a method which produces schedules with small values for $\frac{m_S}{m_{opt}}$, $\frac{c_S}{c_{opt}}$, and κ.

4 The Algorithm

At first we consider a simple non-preemptive *first fit* list scheduling algorithm where the ordering of all jobs is determined by the submission times. This method also allows to take into account those special hardware request or node ranges (with an appropriate strategy) mentioned in Section 2. Although, this algorithm is actually used in commercial schedulers, it may produce bad results as can be seen by the example below.

Example. Simply assume R jobs with $h_i = R$, $r_i = 1$ and R jobs with $h_i = 1$, $r_i = R$. If jobs are submitted in quick succession from both groups alternatively, then $m_S = O(R^2) = R \cdot O(R) = R \cdot m_{opt}$.

Even modifications like backfilling [9] cannot guarantee to avoid such a situation. To solve this problem we introduce preemption and accept $\kappa > 1$. A suitable algorithm for this purpose may be PSRS (Preemptive Smith Ratio Scheduling) [10] which is based on a list and uses gang scheduling. The list order in PSRS is determined by the ratio $\frac{w_i}{r_i h_i}$. As this ratio is the same for all jobs in our scheduling problem (see Definition 3.1) any order can be used and PSRS is able to incorporate FCFS. An adaption of PSRS to our scheduling problem is called PFCFS (Preemptive FCFS) and given in Table 1.

Intuitively, a schedule produced by Algorithm PFCFS can be described as the interleaving of two non-preemptive FIFO schedules where one schedule contains at most one (wide) job at any time instant. Note that only a wide job ($r_i > \frac{R}{2}$) can cause preemption and therefore increase the completion time of a previously submitted job. Further, all jobs are started in FIFO order.

Instruction A is responsible for the on-line character of Algorithm PFCFS. We call any time period Δ a period of available resources (PAR), if during the whole period Instruction A is executed and at least $\frac{R}{2}$ resources are idle. Note that any execution of Algorithm PFCFS will end with a PAR.

5 Theoretical Analysis

Before addressing the bounds of schedules produced by Algorithm PFCFS in detail, we describe a few specific properties of schedules where $w_i = r_i h_i$ holds for all jobs i. Note that these properties can also be easily derived from more general statements of other publications.

COROLLARY 5.1. *Assume that $w_i = h_i$ holds for all jobs i of a sequential job system τ. Then any non-preemptive schedule S with no intermediate idle times between 0 and the last completion time is optimal and there is*

$$c_{opt} = c_S = \frac{1}{2}(\max_{i\in\tau}\{t_i^2\} + \sum_{i\in\tau} h_i^2)$$

Proof. The optimality of schedule S for any order of the jobs is a direct consequence of Smith's rule [13].

The bound is clearly true for $|\tau| = 1$. Adding a new job k to job system τ and executing it directly after the other jobs of schedule S produces schedule S' with

$$\begin{aligned} c_{S'} &= c_S + h_k^2 + h_k \max_{i\in\tau}\{t_i\} \\ &= \frac{1}{2}(\max_{i\in\tau}\{t_i^2\} + \sum_{i\in\tau} h_i^2) + h_k \max_{i\in\tau}\{t_i\} + h_k^2 \\ &= \frac{1}{2}(\max_{i\in\tau}\{(t_i + h_k)^2\} + \sum_{i\in\tau} h_i^2 + h_k^2) \end{aligned}$$

COROLLARY 5.2. *Assume that $w_i = r_i h_i$ holds for all jobs i in a job system τ. Replacing a job i in any non-preemptive schedule S by the successive execution of two jobs i_1 and i_2 with $r_{i_1} = r_{i_2} = r_i$, $h_i = h_{i_1} + h_{i_2}$ and $w_i = w_{i_1} + w_{i_2}$ reduces c_S by $h_{i_1} h_{i_2} r_i$.*

Proof. Splitting of job i has no effect on the contribution to c_S of any other job. It is also independent of the location of job i in the schedule as the weight of i and the sum of the weights of i_1 and i_2 are the same. While the contribution of job i is $r_i h_i^2 + w_i r_i (t_i - h_i)$ in the original schedule, it is $r_i(h_{i_1}^2 + h_{i_2}(h_{i_1} + h_{i_2})) + w_i r_i (t_i - h_i)$ in the new schedule. The proof still holds in the preemptive case if the second job is not preempted.

As already mentioned in the previous section PFCFS will generate non-preemptive FCFS schedules if all jobs are small, i.e. they require at most 50% of the maximum amount of resources ($r_i \le \frac{R}{2}$). First, we restrict ourselves to this case and prove some bounds. In Lemma 5.1 we consider scenarios with a single PAR.

```
        while (the parallel computer is active) {
            if ($Q = \emptyset$ and no new jobs have been submitted)
A               wait for the next job to be submitted;
            attach all newly submitted jobs to $Q$ in FIFO order;
            Pick the first job $i$ of $Q$ and delete it from $Q$;
            if ($r_i \leq \frac{R}{2}$) {
B               wait until $r_i$ resources are available;
                start $i$ immediately;}
            else {
C               wait until less than $r_i$ resources are used;
D               If (job $i$ has not been started)
E_1                 wait until $r_i$ resources are available or time period $h$ has passed;
                else
E_2                 wait until the previously used subset of $r_i$ resources is available
                        or time period $h$ has passed;
                If (the required $r_i$ resources are available)
                    start or resume execution of $i$ immediately;
                else {
F                   preempt all currently running jobs;
                    start or resume execution of $i$ immediately;
G                   wait until $i$ has completed or time period $h$ has passed;
H                   resume the execution of all previously preempted jobs;
                    If (the execution of $i$ has not been completed)
                        Goto $D$; }}}
```

Table 1: The Scheduling Algorithm PFCFS

LEMMA 5.1. *Let $r_i \leq \frac{R}{2}$ and $w_i = r_i \cdot h_i$ for all jobs. Also assume that there is no PAR before the submission of the last job. Then Algorithm PFCFS will only produce non-preemptive schedules S with the properties*

1. $m_S < 3 \cdot m_{opt}$,
2. $c_S < 2 \cdot c_{opt}$, *and*
3. S *is* 1*-fair.*

Proof. As Instructions $C - H$ cannot be executed, Algorithm PFCFS only produces non-preemptive FCFS schedules which are 1-fair.

Note that c_{opt} is lower bounded by the cost of the optimal schedule where the submission times of all jobs are ignored.

We define the time instant $x_1 = \max\{t|$ *for any time instant* $t' < t$ *there are less than* $\frac{R}{2}$ *resources idle*$\}$. Note that $x_1 \geq \max_{i\in\tau}\{s_i\}$.

Next, we are transforming job system τ into job system τ' by splitting each job i with $t_i - h_i < x_1 < t_i$ into two jobs at time x_1. According to Corollary 5.2 we have $c_{S'} = c_S - \delta$ and $c_{opt}(\tau') > c_{opt}(\tau) - \delta$. We partition τ into two disjoint sets τ'_1 and τ'_2 where τ'_2 is the set of all jobs starting at x_1 in S'.

Now, we define another time instant:

$$x_2 = \frac{1}{R}\Big(\sum_{i\in\tau'_1} h_i r_i + \sum_{i\in\tau'_2}(\min\{x_2, h_i\})r_i\Big)$$

In order to produce a worst case schedule we maximize x_1 by making the (impossible) assumption that exactly $\frac{R}{2}$ resources are idle in schedule S at any time instant $t < x_1$. With Corollary 5.1 we then obtain:

$$\begin{aligned}
c_{S'} &= \frac{x_1^2 R}{4} + \frac{1}{2}\sum_{i\in\tau'_1} h_i^2 r_i + \sum_{i\in\tau'_2}(h_i + x_1)h_i r_i \\
c_{opt}(\tau') &\geq \frac{x_2^2 R}{2} + \frac{1}{2}\sum_{i\in\tau'} h_i^2 r_i + \\
&\quad \frac{1}{2}\sum_{i\in\tau'_2}(\max\{h_i - x_2, 0\})^2 r_i + \\
&\quad \sum_{i\in\tau'_2}(\max\{h_i - x_2, 0\})x_2 r_i
\end{aligned}$$

By use of the definition of x_2 and the relations $x_2 \geq \frac{x_1}{2}$ and $\min\{x_2, h_i\} + \max\{h_i - x_2, 0\} = h_i$ for all jobs i this results in

$$2c_{opt}(\tau) \geq 2c_{opt}(\tau') + \delta > c_{S'} + \delta \geq c_S.$$

Note that the bound for $\frac{c_S}{c_{opt}}$ can also be derived by a generalization of Lemma 4.1 in a paper by Turek et al. [15].

For the makespan $m_{opt} \geq h_i$ and $m_{opt} > \frac{t_i - h_i}{2}$ holds for all jobs $i \in \tau$. With j being the job that finishes last in the schedule, we have

$$\frac{m_S}{m_{opt}} = \frac{t_j}{m_{opt}} = \frac{t_j - h_j}{m_{opt}} + \frac{h_j}{m_{opt}} < 3.$$

Note that no job system with jobs in τ_2' can generate a worst case ratio $\frac{c_S}{c_{opt}}$. For the purpose of worst case analysis we can therefore assume that $\tau_2' = \emptyset$. Also, a worst case ratio $\frac{c_S}{c_{opt}}$ requires that $h_i \ll m_S$ for all jobs in order to make the term $\frac{1}{2}\sum_{i\in\tau_1'} h_i^2 r_i$ arbitrary small. Nevertheless, the bounds for PFCFS given in Lemma 5.1 are tight for $1 \ll R$.

Next, we remove the PAR restriction.

THEOREM 5.1. *Let $r_i \leq \frac{R}{2}$ and $w_i = r_i \cdot h_i$ for all jobs. Then Algorithm PFCFS will only produce non-preemptive schedules S with the properties*

1. $m_S < 3 \cdot m_{opt}$,
2. $c_S < 2 \cdot c_{opt}$, *and*
3. *S is 1-fair.*

Proof. The bounds for m_S and c_S are proven by induction in the number of PARs occurring during the execution of Algorithm PFCFS. As shown in Lemma 5.1 the bounds hold for one PAR.

Now, let job i with submission time s_i be the last job ending a PAR. Next, each job j with $s_j < s_i$ and $t_j > 2s_i$ is split at time $2s_i$ into two jobs j_1 and j_2. Note that in the optimal schedule we have $t_j(opt) - s_i > h_{j_2}$. Therefore, we define $s_{j_1} = s_j$ and $s_{j_2} = s_i$. Replacing all those jobs j in S by the corresponding jobs j_1 and j_2 will reduce c_S by some δ.

This way, we have transformed the job system τ into two job systems τ_1' and τ_2' such that $s_j < s_i$ for all $j \in \tau_1'$ and $s_j \geq s_i$ for all $j \in \tau_2'$. We say that $\hat{c}_{opt}(\tau_2')$ are the costs of the optimal weighted completion time schedule of τ_2' when disregarding the submission times for all jobs in τ_2'. Note that the conditions of Lemma 5.1 hold for τ_2'.

We denote by $c_S(\tau_2')$ the part of $c_S(\tau)$ which is attributed to the jobs in τ_2'. As only jobs from τ_2' can use resources in S after $2s_i$, $c_S(\tau_2')$ is smaller than the cost of a PFCFS schedule for τ_2' starting at $2s_i$. Using Lemma 5.1 we therefore obtain

$$c_S(\tau_2') < 2s_i \sum_{i\in\tau_2'} w_i + 2\hat{c}_{opt}(\tau_2').$$

With the induction assumption $c_S(\tau_1') < 2c_{opt}(\tau_1')$ this results in

$$\begin{aligned} c_S - \delta &= c_S(\tau_1') + c_S(\tau_2') \\ &< 2(c_{opt}(\tau_1') + s_i \sum_{i\in\tau_2''} w_i + \hat{c}_{opt}(\tau_2') \\ &< 2c_{opt} - \delta. \end{aligned}$$

Using a similar notation it is easy to see that $m_S(\tau_1') \leq 2s_i$. Therefore, τ_1' can be ignored for determining the makespan. With the help of Lemma 5.1 we finally have

$$m_S < 2s_i + 3\hat{m}_{opt}(\tau_2') < 3m_{opt}.$$

A brief look at the example of the workload for a real machine (see Table 2) shows that wide jobs are rather uncommon. This explains the acceptable performance of FCFS schedules in many cases.

For the general analysis of Algorithm PFCFS we assume that the minimal execution time of any job is h and introduce the relative preemption penalty $\bar{p} = \frac{p}{h}$, see [10]. Without the definition of a minimal application time there is no constant approximation factor for Algorithm PFCFS in general. In real application there is always a minimal application time due to the loading of a job.

In the next lemma we evaluate the degree of machine utilization for the general case.

LEMMA 5.2. *At least $\frac{1}{4+2\bar{p}}$ of the resources are used on average during any time frame of a schedule produced by Algorithm PFCFS if*

- *the time frame does not include any part of a PAR and*
- *the time frame starts and ends during the execution of Instructions A, B, or C.*

The first condition simply prevents underutilization due to a lack of jobs. In addition, the second condition requires that no preemption period is partially included in the selected time frame.

Proof. Without preemption more than 50% of the resources are used at any time instant outside of a PAR. Therefore, we only look at a time frame which includes preemptive execution of jobs. In particular we assume that a wide job i ($r_i > \frac{R}{2}$) is completed during Instruction G and examine the time period Δ from the start of Instruction D to the next execution of Instruction A, B, or C. Δ can be split into 3 parts:

1. a single part of length $h(2+\bar{p})$,
2. $\lfloor\frac{h_i}{h}\rfloor - 1$ parts of length $h(2+\bar{p})$, and
3. a single part of length $(\frac{h_i}{h} - \lfloor\frac{h_i}{h}\rfloor + 1)h(1+\bar{p})$.

Note that each invocation of Instructions E_1 and E_2 is executed for a time period h during Δ. As $h_i \geq h$ a resource-time product of more than Rh is used during the first part of Δ. During the second part of Δ it is possible that a single long running sequential job prevents the availability of the necessary resources to execute the rest of job i in a non-preemptive fashion. Hence, we only know that the resource-time product is more than $\frac{Rh}{2}$ for a part of length $h(2+\bar{p})$. Finally, only a resource-time product of more than $(\frac{h_i}{h} - \lfloor\frac{h_i}{h}\rfloor)\frac{hR}{2}$ is used during the last part of Δ. For the purpose of worst case analysis, we can therefore assume that $\frac{h_i}{h} - \lfloor\frac{h_i}{h}\rfloor \to 0$.

This means that Δ has a length of $h_i(2+\bar{p})+h(1+\bar{p})$ and that a combined resource-time product of more than $(h_i+h)\frac{R}{2}$ is used during Δ.

The average usage of resources can be significantly increased if we allow a version of backfilling [9], that is the earlier execution of small jobs while a wide job is executed in a preemptive fashion. Provided that a sufficient number of those small jobs is available, the average usage of resources will increase to $\frac{3}{6+4\bar{p}}$. However, the completion time of a wide job may be delayed by this method (contrary to the original backfilling suggested by Lifka).

Next, we generalize Lemma 5.1 to the general case.

LEMMA 5.3. *Let $w_i = r_i \cdot h_i$ for all jobs. Also assume that there is no PAR before the submission of the last job. Then Algorithm PFCFS will only produce schedules S with the properties*

1. *$m_S < (4+2\bar{p})m_{opt}$,*
2. *$c_S < (3.562+3.237\bar{p})c_{opt}$, and*
3. *S is $(2+2\bar{p})$-fair.*

Proof. The completion time of a small job ($r_i \leq \frac{R}{2}$) may increase by $h_i(1+2\bar{p})$ due to a wide job which is submitted later and causes preemption. This yields the fairness result.

Based on the proof of Lemma 5.1 we assume that the execution time of all jobs is small compared to the makespan of the schedule. Using our experiences in Lemma 5.1 we also consider for the determination of $\frac{c_S}{c_{opt}}$ only those schedules where the job set τ_2' (of Lemma 5.1) is empty.

Hence, our schedules consist of periods with wide jobs causing preemption and other periods containing only small jobs. W.l.o.g. we can assume that the average usage of the first periods is given by the bound of Lemma 5.2 while during the other periods $\frac{R}{2}$ resources are always used, see the proof of Lemma 5.1.

As the resource usage in the second group of periods is higher and $w_i = h_i r_i$ holds for all jobs, a worst case schedule is generated by scheduling all those periods after the preemptive periods. Of course, there are always a few small jobs which must also be scheduled in the preemptive periods but they are also assumed to be scheduled on top of the preemptive jobs for the purpose of the analysis.

As derived in the proof of Lemma 5.2 a preemptive job i requires in the worst case a period of length $h_i(2+\bar{p}) + h\bar{p} < h_i(2+2\bar{p})$ if those small jobs in the beginning are not taken into account. Note that the second bound is not tight for $h_i \gg h$.

By assuming that in schedule S the sum of the execution times for all preemptive jobs is x_1 and all small jobs require a combined resource-time product of $\frac{x_2 R}{2}$ we obtain the following weighted completion time costs:

$$c_S < \frac{x_1^2 R}{4}(2+2\bar{p}) + \frac{x_2^2 R}{4} + \frac{x_1 x_2 R}{2}(2+2\bar{p})$$

$$c_{opt} \geq \begin{cases} \frac{x_1^2 R}{4} + \frac{x_2^2 R}{4} & \text{for } x_2 \leq x_1 \\ \frac{x_1^2 R}{2} + \frac{(x_2-x_1)x_1 R}{2} + \frac{(x_2-x_1)^2 R}{8} & \text{for } x_2 > x_1 \end{cases}$$

First, we consider the case $x_2 < x_1$. From the inequations we determine a bound for the ratio

$$\frac{c_S}{c_{opt}} < \frac{y^2 + 2y(2+2\bar{p}) + 2 + 2\bar{p}}{y^2+1}.$$

As $\bar{p}$ may be different for each machine, we generate two separate functions $\frac{y^2+4y+2}{y^2+1}$ and $\frac{4y\bar{p}+2\bar{p}}{y^2+1}$ and maximize both individually. This way we obtain

$$\frac{c_S}{c_{opt}} < 3.562 + 3.237\bar{p}.$$

In both cases we have $y = \frac{x_2}{x_1} < 1$. For $x_2 > x_1$ we obtain the following (smaller) bound

$$\frac{c_S}{c_{opt}} < \frac{y^2 + 2y(3+2\bar{p}) + 7 + 6\bar{p}}{\frac{1}{2}y^2 + 2y + 2} \leq 3.5 + 3\bar{p}.$$

Next assume that we have a set $\tau_2' \subset \tau$ of jobs which are scheduled at time $(2+2\bar{p})x_1 + x_2$ in schedule S.

Then those jobs are replaced by small jobs such that the resource time product $\sum_{i\in\tau_2'} h_i r_i$ remains invariant and the small jobs are scheduled using $\frac{R}{2}$ at any time instant. This reduce c_{opt} at least as much as c_S. Therefore, the ratio $\frac{c_S}{c_{opt}}$ cannot increased by $\tau_2' \neq \emptyset$.

Using the same definitions of x_1 and x_2 the following bounds hold for the optimal makespan: $m_{opt} \geq h_i$ for each job i, $m_{opt} \geq x_1$ and $m_{opt} > \frac{x_1+x_2}{2}$. With j being the job that finishes last we obtain

$$\begin{aligned} m_S &\leq h_j + x_2 + x_1(2+2\bar{p}) \\ &= h_j + 2\frac{x_1+x_2}{2} + (1+\bar{p})x_1 \\ &< (4+2\bar{p})m_{opt}. \end{aligned}$$

Finally, we again remove the constraint on the number of PARs.

THEOREM 5.2. *Let $w_i = r_i \cdot h_i$ for all jobs. Then Algorithm PFCFS will only produce schedules S with the properties*

1. $m_S < (4+2\bar{p})m_{opt}$,
2. $c_S < (3.562 + 3.237\bar{p})c_{opt}$, *and*
3. *S is $(2+2\bar{p})$-fair.*

Proof. The proof is done in the same fashion as the proof of Theorem 5.1. There are only some minor differences:

1. If job i with submission time s_i is the last job ending a PAR, then the cut must be positioned between $(3+2\bar{p})s_i$ and $(3.562+3.237\bar{p})s_i$ which gives enough freedom to choose the cut at a time instant when the execution of small jobs is resumed after a preemption penalty.

2. There is an extension to Corollary 5.2 for preemptive schedules. However, if a split reduces c_S by δ, then c_{opt} may only be reduced by $\frac{\delta}{x}$ if the difference between the starting time and the completion time of any job i is bounded by xh_i.

Taking these changes into account the techniques of the proof for Theorem 5.1 also prove the statements of this theorem.

As mentioned before the bounds for both ratios $\frac{c_S}{c_{opt}}$ and $\frac{m_S}{m_{opt}}$ can be reduced if backfilling is used. However, this will result in a more complicated analysis.

6 Experimental Analysis

In this section we evaluate various forms of preemptive FCFS scheduling with the help of workload data from the CTC SP2 for the months July 1996 to May 1997. These data include all batch jobs which ran on the CTC SP2 during this time frame. For each job the submission time, the start time, and the completion time was recorded. The submission queue was also provided as well as requests for special hardware for some jobs. The reasons for the termination of a job (successful completion, failure to complete, user termination, termination due to exceeding of the queue limit time) were not given and are not relevant for our simulations. The generation of the trace data had no influence on the execution time of the individual jobs.

The CTC uses a batch partition consisting of 430 nodes. But there are only few jobs which need more than 215 nodes. Taking further into account the execution times of these jobs, preemption will not produce any noticeable gain (see Theorem 5.1). Therefore, we assumed for our experiments parallel computers with 128 and 256 nodes, respectively. All jobs requiring more nodes were simply removed. A list of the number of jobs for each month is given in Table 2.

	Total	$r_i \leq 256$	$r_i \leq 128$
Jul 96	7953	99.75%	99.30%
Aug 96	7302	99.69%	99.07%
Sep 96	6188	99.87%	98.67%
Oct 96	7288	99.85%	99.75%
Nov 96	7849	99.90%	99.58%
Dec 96	7900	99.91%	99.85%
Jan 96	7544	99.92%	99.50%
Feb 96	8188	99.87%	99.65%
Mar 96	6945	99.83%	99.48%
Apr 96	6118	99.74%	99.46%
May 96	5992	99.87%	99.50%

Table 2: Number of Jobs

In the experiments a wide job only preempts enough currently running jobs to generate a sufficiently large partition. Those small jobs are selected by use of a simple greedy strategy. This modification of Algorithm PFCFS does not affect the theoretical analysis but improves the schedule performance for real workloads significantly.

We generated our own FCFS schedule as a reference schedule and did not use the CTC schedule as some jobs were for instance submitted in October and started in November. Using the CTC schedule would not allow to evaluate each month separately. As the preemption penalty for the IBM gang scheduler is less than a millisecond it is neglected ($\bar{p} = 0$).

We did a large number of simulations with different preemption strategies. For each strategy and each month we determined the makespan, the total weighted completion time, and the total weighted flow time and compared these values with the results of a simple non-preemptive FCFS schedule. In Table 3 we describe three of the preemption strategies tested. Experimental results for these strategies are given in Tables 4 to 9 in the Appendix.

$PFCFS_1$	A wide job causes preemption after it has been waiting for at least 10 min. Then the two gangs are preempted every 10 min until one gang becomes empty.
$PFCFS_2$	A wide job causes preemption after it has been waiting for at least 1 min. After running for 1 min the wide job is preempted and the previously preempted job are resumed. Finally, the wide job waits until the partition becomes empty (no further preemption).
$PFCFS_3$	A wide job causes preemption after it has been waiting for at least 10 min. Then it runs to completion. Afterwards, the preempted jobs are resumed.

Table 3: Different Preemptive Strategies

We noticed that the schedule quality is very sensitive to the preemption strategy. While significant improvements over FCFS are possible, FCFS may still outperform some preemptive methods. Although there may be general explanations for some phenomena (short execution times of wide jobs which fail to run due to a bug), the workload certainly plays a significant role. This can be best demonstrated by looking at the total weighted flow time where strategy $PFCFS_2$ produced almost 50% improvement over FCFS for November 1996 while it performed more than 9% worse than FCFS in January 1997, see Table 6 in the Appendix.

From these results we conclude that using a preemptive FCFS strategy is potentially beneficial in parallel job scheduling. But the optimal parameter setting depends on the workload and must be carefully selected. We suggest that the parameters are determined on-line in an adaptive fashion based on the workload of previous months.

Acknowledgment The authors are especially grateful to Steve Hotovy for providing them with the workload data of the CTC SP2.

References

[1] S. Chakrabarti, C. Phillips, A.S. Schulz, D.B. Shmoys, C. Stein, and J. Wein. Improved approximation algorithms for minsum criteria. In *Proceedings of the 1996 International Colloquium on Automata, Languages and Programming*. Springer Verlag Lecture Notes in Computer Science, 1996.

[2] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *Proceedings of the 7th SIAM Symposium on Discrete Algorithms*, pages 159–167, January 1996.

[3] D.G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860. In D.G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 337–360. Springer–Verlag, Lecture Notes in Computer Science 949, 1995.

[4] D.G. Feitelson and L. Rudolph. Parallel job scheduling: Issues and approaches. In D.G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 1–18. Springer–Verlag, Lecture Notes in Computer Science 949, 1995.

[5] D.G. Feitelson and L. Rudolph. Towards convergence in job schedulers for parallel supercomputers. In D.G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer–Verlag, Lecture Notes in Computer Science 1162, 1996.

[6] A. Feldmann, J. Sgall, and S.-H. Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science*, 130:49–72, 1994.

[7] S. Hotovy. Workload evolution on the cornell theory center ibm sp2. In D.G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 27–40. Springer–Verlag, Lecture Notes in Computer Science 1162, 1996.

[8] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the 29th ACM Symposium on the Theory of Computing*, 1997.

[9] D.A. Lifka. The anl/ibm sp scheduling system. In D.G. Feitelson and L. Rudolph, editors, *IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 295–303. Springer–Verlag, Lecture Notes in Computer Science 949, 1995.

[10] U. Schwiegelshohn. Preemptive weighted completion time scheduling of parallel jobs. In *Proceedings of the 4th Annual European Symposium on Algorithms (ESA96)*, pages 39–51. Springer Verlag Lecture Notes in Computer Science LNCS 1136, September 1996.

[11] U. Schwiegelshohn, W. Ludwig, J.L. Wolf, J.J. Turek, and P. Yu. Smart SMART bounds for weighted response time scheduling. *SIAM Journal on Computing.* Accepted for publication.

[12] D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6):1313–1331, December 1995.

[13] W. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[14] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Preprint*, 1996.

[15] J.J. Turek, W. Ludwig, J.L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. Yu. Scheduling parallelizable 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.

[16] F. Wang, H. Franke, M. Papefthymiou, P. Pattnaik, L. Rudolph, and M.S. Squillante. A gang scheduling design for multiprogrammed parallel computing environments. In D.G. Feitelson and L. Rudolph, editors, *IPPS'96 Workshop: Job Scheduling Strategies for Parallel Processing*, pages 111–125. Springer–Verlag, Lecture Notes in Computer Science 1162, 1996.

Appendix

	$PFCFS_1$	$PFCFS_2$	$PFCFS_3$
Jul 96	-3.5%	-16.3%	+20.0%
Aug 96	-8.3%	-22.2%	+14.3%
Sep 96	-10.0%	-25.4%	+8.7%
Oct 96	-4.9%	-23.2%	+12.2%
Nov 96	-18.0%	-30.0%	+11.2%
Dec 96	-12.5%	-22.3%	+20.3%
Jan 97	-4.9%	-16.9%	+16.4%
Feb 97	-4.1%	-15.4%	+32.3%
Mar 97	-5.6%	-14.1%	+15.4%
Apr 97	-14.3%	-26.7%	+8.6%
May 97	-15.8%	-29.2%	+3.0%
Sum	-9.5%	-22.1%	+14.4%

Table 4: Results for Makespan and 128 Nodes

	$PFCFS_1$	$PFCFS_2$	$PFCFS_3$
Jul 96	-7.1%	-19.7%	+13.8%
Aug 96	-6.1%	-20.4%	+13.2%
Sep 96	-10.9%	-25.5%	+6.5%
Oct 96	-3.3%	-20.9%	+11.5%
Nov 96	-21.0%	-33.4%	+5.7%
Dec 96	-15.0%	-25.6%	+20.7%
Jan 97	-4.6%	-13.4%	+12.1%
Feb 97	-5.8%	-18.3%	+25.0%
Mar 97	-5.3%	-14.2%	+16.7%
Apr 97	-11.8%	-24.5%	+11.8%
May 97	-14.0%	-28.4%	+2.7%
Sum	-9.6%	-22.2%	+12.3%

Table 5: Results for Completion Time and 128 Nodes

	$PFCFS_1$	$PFCFS_2$	$PFCFS_3$
Jul 96	-10.4%	-30.6%	+22.0%
Aug 96	-22.6%	-41.9%	+2.0%
Sep 96	-15.9%	-37.3%	+9.4%
Oct 96	-5.2%	-32.9%	+17.4%
Nov 96	-29.7%	-47.3%	+8.1%
Dec 96	-22.4%	-39.1%	+31.7%
Jan 97	+28.1%	+9.6%	+63.0%
Feb 97	-8.2%	-28.8%	+38.3%
Mar 97	-8.4%	-22.3%	+26.2%
Apr 97	-18.4%	-38.3%	+18.4%
May 97	-19.5%	-39.5%	+3.7%
Sum	-13.6%	-33.2%	+19.1%

Table 6: Results for Flow Time and 128 Nodes

	$PFCFS_1$	$PFCFS_2$	$PFCFS_3$
Jul 96	-5.2%	-6.9%	+13.3%
Aug 96	-2.4%	-9.0%	+11.7%
Sep 96	-8.1%	-19.6%	+18.2%
Oct 96	-23.5%	-28.7%	-6.5%
Nov 96	-0.5%	-9.6%	+24.2%
Dec 96	-0.6%	-5.6%	+18.9%
Jan 97	-11.9%	-18.7%	+12.1%
Feb 97	-8.5%	-13.7%	+29.6%
Mar 97	-2.4%	-7.3%	+16.1%
Apr 97	-1.6%	-8.6%	+24.3%
May 97	+1.4%	-9.1%	+24.4%
Sum	-6.4%	-13.0%	+16.2%

Table 7: Results for Makespan and 256 Nodes

	$PFCFS_1$	$PFCFS_2$	$PFCFS_3$
Jul 96	-9.3%	-11.6%	+12.4%
Aug 96	-1.5%	-8.1%	+8.1%
Sep 96	-9.4%	-19.4%	+13.8%
Oct 96	+0.9%	-4.5%	+20.7%
Nov 96	-0.7%	-8.5%	+23.1%
Dec 96	0.0%	-6.2%	+19.5%
Jan 97	-11.2%	-15.9%	+2.3%
Feb 97	-11.9%	-15.1%	+7.9%
Mar 97	-4.1%	-7.5%	+11.6%
Apr 97	+0.9%	-6.2%	+22.9%
May 97	+3.2%	-7.2%	+24.0%
Sum	-4.2%	-10.4%	+14.6%

Table 8: Results for Completion Time and 256 Nodes

	$PFCFS_1$	$PFCFS_2$	$PFCFS_3$
Jul 96	-37.2%	-46.8%	+49.8%
Aug 96	-6.9%	-36.5%	+36.9%
Sep 96	-24.9%	-51.9%	+35.5%
Oct 96	+4.7%	-22.0%	+190.6%
Nov 96	-3.2%	-31.8%	+86.1%
Dec 96	-2.2%	-30.1%	+81.2%
Jan 97	-30.7%	-42.3%	+6.9%
Feb 97	-36.2%	-47.3%	+23.7%
Mar 97	-14.4%	-28.0%	+44.2%
Apr 97	+5.3%	-41.3%	+147.3%
May 97	+9.8%	-22.5%	+73.3%
Sum	-16.1%	-38.0%	+55.6%

Table 9: Results for Flow Time and 256 Nodes