Scheduling Against an Adversarial Network

Stefano Leonardi * Alberto Marchetti-Spaccamela*

Friedhelm Meyer auf der Heide †

ABSTRACT

Using idle times of the processors is a well-known approach to run coarse grained parallel algorithms for extremely complex problems. We present on-line algorithms for scheduling the processes of a parallel application that is known off-line on a dynamic network in which the idle times of the processors are dictated by an adversary. We also take communication and synchronization costs into account.

Our first contribution consists of a formal model to restrict the adversary in a reasonable way. We then show a constant factor approximation for the off-line scheduling problem. As this problem has to take communication cost into account, it can be seen as a generalization of many NP-hard parallel machine scheduling problems. Finally, we present on-line algorithms for different models with constant or with "nearly constant" competitive ratio.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: performance attributes; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical algorithms and problems—*sequencing and scheduling*

General Terms Algorithms, Performance, Theory

Keywords

Parallel machine scheduling, Competitive Algorithms, Parallel computation

1. INTRODUCTION

Using idle times of the processors of a LAN or WAN is a well-known approach to run coarse grained parallel algorithms for extremely complex problems. This approach is for example used for many number theoretic algorithms, or, as one of the most prominent examples, SETI@home [23] where thousands of Internet-connected computers are used in the search for extra-terrestrial intelligence. Grid computing, i.e., sharing resources distributed in, for example, the Internet in order to run complex applications can be viewed as an extension of this scenario, because now nodes with very different capabilities are considered. The system therefore is highly heterogeneous.

In this paper we focus on the first scenario. We consider a system of identical processors that are used by their "owners" who only allow idle times of their machines to be used for the parallel application. Thus, from the point of view of our parallel algorithm, the network is highly dynamic and its future behavior can not be foreseen. Therefore processes produced by our program have to be scheduled on this dynamic network in an on-line fashion.

We present an on-line algorithm for scheduling a program on such a dynamic network. Our programs are described by an evolving tree of processes of possibly different execution times. In contrast to most scheduling algorithms, in our case the program in known off-line, whereas the idle times of the processors are dictated by an adversary. It is important to note that we also take communication into account by assuming some cost g for a migration of a process, and demanding a synchronisation between two consecutive levels of the tree. Thus our model is inspired by the Valiant's BSP paradigm for parallel computation.

It is easily seen that an unrestricted adversary that may dictate the distribution of idle times in the processors can enforce unbounded worst case performance for the

*Dipartimento di Informatica e Sistemistica. Università di Roma "La Sapienza". leon@dis.uniroma1.it. Partially supported by the Future and Emerging Technologies programme of the EU under contracts number IST-2001-33555 COSIN "Co-evolution and Self-organization in Dynamical Network", EU Contract 001907 DELIS "Dynamically Evolving, Large Scale Information Systems", and by the Italian research project ALINWEB: "Algoritmica per Internet e per il Web", MIUR – Programmi di Ricerca di Rilevante Interesse Nazionale.

†Heinz Nixdorf Institute and Computer Science Department. University of Paderborn. fmadh@upb.de. Partially supported by the IST Programme of the EU under contract 001907 DELIS "Dynamically Evolving, Large Scale Information Systems" and by DFG-SFB 376 "Massively Parallel Computing".

SPAA'04, June 27–30, 2004, Barcelona, Spain.

algorithm. On the other hand, typical systems do not change the idle times reserved for an application in such a drastic way. Therefore, a suitable model for the dynamics of the network, i.e., a reasonable, realistic restriction of the adversary is necessary.

In this paper, we contribute the following:

- We present two models for restricting the adversary in a reasonable way. To our knowledge, this is the first formal model for such a behavior.
- We show a constant factor approximation for the off-line scheduling problem. As this problem has to take communication cost into account, it can be seen as a generalization of many NP-hard scheduling problems.
- Based on this, we present on-line algorithms for different models with constant or with "nearly constant" competitive ratio.

The next three subsections present our model, our new results, and related results. Section 2 then describes the off-line approximation. Section 3 presents our on-line algorithms.

1.1 The model

The program is a leveled tree. The nodes of the tree are processes. Each process p has execution time w(p). The computation proceeds in *supersteps*. In Superstep t, all processes from level t are processed, superstep $t+1$ can start only if all processes from level t have been processed.

The machine consists of n processors $Q_1, \ldots, Q_n$. The machine runs in synchronous phases.

Executing a program is done superstep by superstep. Superstep 0 starts with the root executed in Q_1. Afterwards, the children of the root are mapped by the scheduler to processors. Superstep t starts when all processes from level t have been mapped to processors. Executing a superstep proceeds in (one or more) synchonized *phases*, each consisting of a computation, a migration and a synchronization. During the computation, every processor executes (some of) the processes mapped to it during the previous migration. Afterwards, those processes of the superstep that are not yet processed are migrated by the scheduler. Finally a global synchronization takes place. In the last phase of superstep t, when all its processes have been completed, the scheduler maps the processes from level $t+1$ to the processors.

The cost of a phase consists of the time T needed for its computation, time L for the synchronization and the cost of the migration. This is given by the maximum number of processes that are migrated from or to a specific processor times an individual migration cost of g. More formally, assume x_i processes are migrated to processor Q_i, and y_i processors are migrated from processor Q_i, at the beginning of a phase. The migration cost of processor Q_i will therefore be equal to $g \times (x_i + y_i)$. The cost of the migration phase is equal to $\max_i g \times (x_i + y_i)$. We will refer to $\max_i(x_i + y_i)$ as the *congestion or the maximum load of the phase*. The cost for the superstep is the sum of the costs of its phases.

The processors of our machine do only supply computation power that varies over time, i.e. idle times to process processes of our leveled tree. In our model we simplify this behavior by assuming that a processor is either on or off at a given time, thus it offers a sequence of disjoint time intervals in which it is available. We approximate the lengths of the processes to powers of two.

More formally, the cost of computation of a superstep t is defined as follows.

The computation is synchronized at the beginning of each phase. For every Q_i, the cost of computation $CC(i,l,t)$ of phase $l = 1, \ldots, l_t$, of superstep t is equal to its length. The congestion of Q_i in phase l of superstep t is denoted by $MC(i,l,t)$. Recall that it is equal to the number of processes migrated from Q_i plus the number of processes migrated to Q_i. The overall cost of phase l of superstep t is $C(l,t) = \max_i CC(i,l,t) + g * \max_i MC(i,l,t) + L$, where L is the cost for synchronisation. The cost of a superstep t is $C(t) = \sum_{l=1}^{l_t} C(l,t)$ The total cost of computation is $C = \sum_t C(t)$. The goal is to find a scheduler that minimizes C.

We denote by C^{OPT} the cost of an optimal off-line schedule. It is clear that such a schedule will only use one phase per superstep.

In the on-line version, the program is known beforehand, while the number of intervals available for scheduling processes is not known beforehand. However, the rate of change of the number of intervals that are available in each phase is constrained.

Consider two time intervals I_1 and I_2 of length T, where T is bigger than a constant valued D. Denote by $q_{i,k}$ and $q'_{i,k}$ the number of maximal intervals of length 2^k in I_1 and I_2 , $k \in \{0, \ldots, K\}$, where machine Q_i is available without interruption.

We consider two restrictions of our adversary. Both are motivated by the insight that the amount of computation power offered by the network obeys some "approximate periodicity", i.e., it does not change dramatically between I_1 and I_2. The stronger restriction demands this individually for each processor, the weaker only for the overall offered computation power of large groups of processors. Let $0 < \epsilon, \alpha < 1$ be constants.

Strong restriction:

$$\mid q_{i,k} - q'_{i,k} \mid \leq \varepsilon q_{i,k} \ \textit{for all } i.$$

Weak restriction:

$$\sum_{i \in J} \mid q_{i,k} - q'_{i,k} \mid \leq \varepsilon \sum_{i \in J} q_{i,k} \ \textit{for all } J \textit{ with } \mid J \mid \geq \alpha n.$$

1.2 New results

In Section 2 we present an on-line algorithm that, based on the log-star paradigm (see [18], [13]), [12]) executes one superstep consisting of m uniform jobs with competitive ratio $O(\log^*(m))$ in case of the weak restriction of the adversary.

In case of the strong restriction of the adversary and non-uniform job sizes, i.e., if different lengths of processes are allowed, we also present in Section 2 a simple scheduling strategy with constant competitive ratio that uses a limited amount of resource augmentation [14]. Our solution is based on the study of the off-line problem given presented in Section 3, where we give a constant factor approximation algorithm to schedule one superstep when

the lengths of the processes and of the intervals are restricted to be powers of 2.

We also prove that this problem is already NP-complete in this specific case. The bound on communication cost, i.e. on the maximum number of processes that can be assigned to a single machine, is therefore in this case the real source of NP-hardness. This problem then generalizes several well known NP-hard parallel machine scheduling problems, which makes the off-line approximation an interesting contribution on its own. A limited amount of resource augmentation allows to drop the assumption that the numbers that are involved are powers of 2.

1.3 Related results

To our knowledge, there are no models and formal analyses known for scheduling processes on a dynamic systems as considered in our paper. On the other hand, many practical projects use the computation power of big networks of "used" processors.

First, there are programs for solving specialized problems, Seti@Home (search for extraterrestrial intelligence, [23]), distributed.net (e.g. breaking cryptographic keys) [9] and Folding@Home (simulation of protein folding, [10]) are some of the most famous ones. These systems work with a large data space, divide this data into small pieces, send them as jobs to the clients, and wait for the results. There is no communication during the computation. If a client does not return the result, a timeout occurs and the job is scheduled to another client. These systems are very powerful in terms of computation power because of the huge number of clients taking part. On the other hand, they are mainly used for algorithms that, in our terminology, consist of a small, constant number of supersteps, each consisting of a huge amount of heavy processes.

Second, there are job scheduling and migration systems. Here one can start jobs, either directly or using a batch system. These jobs are executed on idle nodes and may migrate to other nodes if the load in the system changes. The Condor system [17] is one example for such system implemented in user space, MOSIX [3] is an extension for the Linux kernel. An overview of some migration systems can be found in [19]. An implementation for migration of the single threads, even in a heterogeneous network with different CPU types, is proposed by Dimitrov and Rego in [8]. They extend the C language and implement a preprocessor which automatically inserts code for keeping track of all local data. Additionally the state in each function is stored, i.e. the actual execution point, and at the start of a function a switch statement is inserted which branches to the right position depending on the state if a thread is recreated. The changes lead to an overhead of 61% in their example, due mainly to the indirect addressing of local variables.

All these systems support only sequential jobs (Condor can schedule parallel jobs using MPI or PVM, but these jobs do not take part in load balancing and migrations; the MOSIX team is working on migratable sockets but there is no implementation available yet). There is some work about checkpointing threaded processes [7] useful to migrate parallel programs for symmetric multiprocessor (SMP) machines, but as far as we know there is no system for distributed parallel applications that supports direct communications between the jobs.

In the project Bayanihan ([22]), a Java-based BSP implementation for volunteer computing systems, the computational work of the BSP supersteps is distributed by a master to the workers. Each worker executes the job and sends the new context and all messages for other processors back to master. Further work is going on to extend the Paderborn University BSP Library (PUB library, see [6], [5], [21]), so that idle times of a LAN or WAN can be used to execute a BSP program efficiently.

Parallel machine scheduling problems with features similar to those encountered in our scenario have been considered in several works. Migration in parallel machine scheduling has been for instance considered in [2, 4, 16, 15]. However, at the best of our knowledge, either the overhead of migration is not considered in the objective function or it is not considered to perform migration in parallel on the different processors. On-line algorithms for parallel machine scheduling in presence of machines breakdown have for instance been considered in [1].

2. THE ON-LINE SCHEDULE

Our idea is to monitor the status of the machines to determine a lower bound T_t on the computational time of the optimum in a generic superstep t. We assume that the optimum has completed superstep $t-1$ not later than s_t. We initialize $T_t = g$. The subroutine *schedule* provided in Section 3 is able to check whether it is feasible to schedule the processes of supertep t in time interval $I_t = [s_t, s_t+T_t)$ and congestion $C_t = T_t/g$. If the answer is negative, we double the estimation T_t. If the answer is positive, we have determined a lower bound $opt_t \geq T_t/2 + L$ on the optimal cost in superstep t. We will therefore set $s_{t+1} = s_t+T_t/2+L$ as a new lower bound on the optimum for the completion of superstep t.

We restrict the length of the processes and the length of the intervals to be a power of 2. Let m_k be the total number of processes of length 2^k to be assigned in superstep t , $0 \leq k \leq K$. In case of uniform processes, let m be the total number of processes to be assigned in superstep t.

Assume from now that a generic superstep is processed by the off-line algorithm in time interval $I = [T, T+s)$. Denote by $l_{i,k}$ the number processes of size 2^k that are assigned on machine $Q_i, \sum_{i=1}^{n} l_{i,k} = m_k$, $0 \leq k \leq K$. In case of uniform processes, l_i processes are assigned to $Q_i, \sum_{i=1}^{n} l_i = m$. Let $S = max\{s, D\}$.

LEMMA 1. *There is a constant c such that each interval I' of length $c \cdot S$ has the following property:*

a) In case of non-uniform processes and the strong restriction: Each Q_i can accommodate $l_{i,k}$ processes of size 2^k also in I'.

b) In case of uniform processes and the weak restriction: Each Q_i can accommodate l'_i processes such that

- $\sum l'_i = 4m$
- $l'_i \leq c \cdot (l_i + \frac{m}{n})$.

PROOF. For proving part a), consider an interval $\tilde{I}$ of size S. As $S \geq D$, the strong restriction guarantees that $(1-\varepsilon)l_{i,k}$ slots for processes of size k are available in Q_i. Thus, if I' consists of $c = \left\lceil \frac{1}{1-\varepsilon} \right\rceil$ consecutive intervals of size s, part a) of the Lemma follows.

In order to prove part b) we apply the following lemma.

LEMMA 2. *Let* $q_1, \ldots, q_1, q'_1, \ldots, q'_n$ *be integers* $\geq 0, 0 < \varepsilon, \alpha < 1$. *Assume that*

$$\sum_{i \in J} |q'_i - q_i| \leq \varepsilon \sum_{i \in J} q_i \text{ for all } J \subseteq \{1, \ldots, n\} \text{ with } |J| \geq \alpha n.$$

Then, for arbitrary $l_1, \ldots, l_n, m \geq 0$ *with* $\sum_{i=1}^n l_i = m$ *and* $l_i \geq \frac{1}{2}\frac{m}{n}$, *it holds that*

$$\begin{aligned} \sum_{i=1}^n min(q'_i, l_i) &\geq \gamma \sum_{i=1}^n min(q_i, l_i), \text{ for } \gamma \\ &= min(1-\varepsilon, \frac{1}{2}(1-\alpha)). \end{aligned}$$

PROOF. Note that it suffices to prove the result for the case that $q'_i \leq q_i$ for all i. In this case, the precondition of the Lemma is equivalent to

$$\sum_{i \in J} q'_i \geq (1-\varepsilon) \sum_{i \in J} q_i \text{ for all } J \subseteq \{1, \ldots, n\} \text{ with } |J| \geq \alpha n.$$

Now let $A = \{i, q'_i \leq q_i \leq l_i\}, U = \{i, q'_i \leq l_i < q_i\}$,and $B = \{i, l_i < q'_i \leq q_i\}$.

If $|B| \geq (1-\alpha)n$ then

$$\begin{aligned} \sum_{i=1}^n \min(q'_i, l_i) &\geq \sum_{i \in B} \min(q'_i, l_i) = \sum_{i \in B} l_i \geq |B| \frac{1}{2}\frac{m}{n} \\ &\geq \frac{1}{2}(1-\alpha) \sum_{i=1}^n \min(q_i, l_i). \end{aligned}$$

If $|B| < (1-\alpha)n$ then $|A \cup U| \geq \alpha n$. Therefore,

$$\sum_{i=1}^n \min(q'_i, l_i) = \sum_{i \in A \cup U} q'_i + \sum_{i \in B} l_i \geq (1-\varepsilon) \sum_{i \in A \cup U} q_i + \sum_{i \in B} l_i$$

$$\geq (1-\varepsilon) \sum_{i \in A} q_i + (1-\varepsilon) \sum_{i \in U} l_i + \sum_{i \in B} l_i \geq (1-\varepsilon) \sum_{i=1}^n \min(q_i, l_i).$$

□

Now consider an arbitrary interval $\tilde{I}$ of size S. Let I ($\tilde{I}$) offer $q_i(q'_i)$ slots for uniform processes in $Q_i, i = 1, \ldots, n$. Then $l_i \leq q_i$. Let $l''_i = l_i + \frac{m}{n}$, $\tilde{l_i} = min(q'_i, l''_i)$. Then $\sum_{i=1}^n l''_i = 2m$ and each $l''_i \geq \frac{m}{n} = \frac{1}{2}\frac{2m}{n}$. As $S \geq D$, the precondition of the above lemma holds and we get:

$$\begin{aligned} \sum_{i=1}^n \tilde{l_i} &= \sum_{i=1}^n min(q'_i, l''_i) \geq \gamma \sum_{i=1}^n min(q_i, l''_i) \\ &\geq \gamma \sum_{i=1}^n min(q_i, l_i) = \gamma \sum_{i=1}^n l_i = \gamma \cdot m. \end{aligned}$$

As, by construction, $\tilde{l_i} \leq l_i + \frac{m}{n}$, we can choose I' to have length $c \cdot s$ for $c = 4\left\lceil \frac{1}{\gamma} \right\rceil$ to obtain part b) of the lemma.

2.1 Uniform processes, weak restriction

Now let us concentrate on uniform processes and the weak restriction. Consider a superstep consisting of m uniform processes that can be processed by the optimal algorithm in interval $I = [T, T+s)$, including migration and synchronization. Our off-line approximation for uniform jobs (compare the last paragraph of Section 2) then produces a schedule that, starting at time T, needs s computation steps, so that Q_i processes l_i jobs with $\sum l_i = m$, and maximum load $C = max\{l_i\} \leq \frac{s}{g}$ suffices.

For our on-line algorithm, we need a sequence of consecutive time intervals $I_1, I_2, \ldots$, such that each of them can accommodate l'_i jobs in Q_i with $\sum l'_i = 4m$, can migrate and receive $l'_i \leq c \cdot (l_i + \frac{m}{n})$ jobs from and to Q_i, and finally synchronizes.

By part b) of Lemma 6, $c \cdot S$ steps suffice for the computation, for $S = max\{s, D\}$. By the above, time $g \cdot c \cdot max(l_i + \frac{m}{n}) \leq 2c \cdot S$ suffices for migration, and $L \leq S$ for synchronisation. Thus, intervals of size $(3c+1) \cdot S$ suffice.

Assume that at the beginning of interval I_l, $\frac{m}{r}$ of the jobs are not yet processed. (Initially, $r = 1$.)

Now we apply the idea of the log*-process, see [18], [13], [12] . We distribute $r \cdot 2c$ copies of each of the $\frac{m}{r}$ processes randomly among the processors, so that Q_i gets $c \cdot (l_i + \frac{m}{n})$ copies. (Note: $\sum_{i=1}^n c \cdot (l_i + \frac{m}{n}) = 2c \cdot m$). Each Q_i sorts its copies randomly.

Let $\left\{X_1, \ldots, X_{\frac{m}{r}}\right\}$ be the 0-1 random variables indicating that the i'th process is not processed in interval I_l. Then $X := \sum_{i=1}^{\frac{m}{r}} X_i$ indicates the number of left-over processes after I_l. As $4m$ out of the $2c \cdot m$ copies will be processed during the interval, we have:

$$\begin{aligned} Prob(X_i = 1) &= \frac{2(c-2)m}{2cm} \cdot \frac{2(c-2)m-1}{2cm-1} \\ &\quad \cdots \frac{2(c-2)m-2cr+1}{2cm-2cr+1} \leq \gamma^r, \end{aligned}$$

for $\gamma = (\frac{c-2}{c})^{2c} \leq \frac{1}{8}$.

Therefore,

$$E(X) \leq \frac{m}{r}(\frac{1}{8})^r.$$

We now aim at a tail estimate for X. For this we apply an extension of the Chernoff Bound from [20]. Consider a family $\{X_1, \ldots, X_q\}$ of 0-1 random variables. This family is called *self-weakening*, if for each $J \subseteq \{1, ..., q\}$, it holds that

$$Prob(X_i = 1 \text{ for all } i \in J) \leq \prod_{i \in J} Prob(X_i = 1).$$

THEOREM 3. *[20] Consider a self-weakening family* $\{X_1, \ldots, X_q\}$ *of 0-1 random variables,* $X := \sum_{i=1}^n X_i$, $E(X) \leq \mu$. *Then,*

$$Prob(X \geq (1+\epsilon) \cdot \mu) \leq (\frac{e^\epsilon}{(1+\epsilon)^{(1+\epsilon)}})^\mu.$$

In order to see that our family $\left\{X_1, \ldots, X_{\frac{m}{r}}\right\}$ is self weakening, consider some $J \subseteq \{1, \ldots, \frac{m}{r}\}$. Then,

$$\begin{aligned} & Prob(X_i = 1 \text{ for all } i \in J) \\ \leq & \frac{2(c-2)m}{2cm} \cdot \frac{2(c-2)m-1}{2cm-1} \cdots \frac{2(c-2)m-2cr|I|+1}{2cm-2cr|I|+1} \\ \leq & [\frac{2(c-2)m}{2cm} \cdot \frac{2(c-2)m-1}{2cm-1} \cdots \frac{2(c-2)m-2cr+1}{2cm-2cr+1}]^{|I|} \\ = & \prod_{i \in I} Prob(X_i = 1). \end{aligned}$$

Therefore, the Theorem yields for the initial phase with $r = 1$ that

$$Prob(X \geq \frac{1}{4}m) = Prob(X \geq 2 \cdot \frac{1}{8}m) \leq (\frac{e}{4})^{\frac{1}{8}m}.$$

For $r \geq 4$ the Theorem yields that

$$Prob(X \geq m(\frac{1}{8})^r) = Prob(X \geq r \cdot \frac{m}{r}(\frac{1}{8})^r)) \leq (\frac{1}{2})^{\frac{m}{r}(\frac{1}{8})^r}$$

.

This is a good bound for $r = o(\log m)$. For larger r, we simply apply the Markov Inequality:

$$Prob(X \geq m(\frac{1}{4})^r) = Prob(X \geq r2^r \cdot \frac{m}{r}(\frac{1}{8})^r) \leq \frac{1}{r2^r}.$$

As $r = \Omega(\log m)$, the above probability is at most $\frac{1}{m^\beta}$, for suitable $\beta > 0$. On average two further phases suffice to finish all jobs. The Markov Inequality also yields a $\frac{1}{m^\beta}$ bound on these probabilities. Therefore we get:

$$Prob(X \geq m(\frac{1}{4})^r) \leq \frac{1}{m^\beta} \text{ for arbitrary } r.$$

Thus, we reduce the number of not yet processed jobs from $\frac{m}{r}$ to $\frac{m}{4^r}$ with probability at least $1 - \frac{1}{m^\beta}$, for arbitrary r. This results in a process that needs expected $O(\log^*(m))$ intervals of lengths $O(max\{s, D\})$. As each interval successfully decreases the number of left-over processes appropriately with probability at least $1 - \frac{1}{m^\beta}$, we achieve *high probability*, i.e. probability $1 - \frac{1}{m^b}$, for arbitrary $b > 0$, if we allow by a suitable constant factor more intervals.

This result implies the following theorem.

THEOREM 4. *Consider a program consisting of t^* supersteps with m_t processes in the t'th superstep, $1 \leq t \leq t^*$, so that the off-line algorithm needs at least D steps for each of at least a constant fraction of these supersteps. Then, in the case of the weak restriction and uniform processes, the on-line scheduling strategy is $O(\log^*(max_t\{m_t\}))$ competitive, with high probability.*

2.2 Non-uniform processes, strong restriction

Now let us concentrate on the case of non-uniform processes and strong restriction. Consider a superstep t in which m_k processes of length 2^k are processed by the off-line algorithm in interval $I = [T, T + s)$, by assigning $l^*_{i,k}$ processes to Q_i, $\sum l^*_{i,k} = m_k$. Now consider a time interval I' of length $c \cdot S$, $S \geq D$. Part a) of Lemma 1 ensures that, in each I', $l^*_{i,k}$ processes can be assigned to Q_i.

The procedure *schedule* presented in Section 3 returns a schedule that spans over at most three time intervals identical to $I_t = [s_t, s_t + T_t)$, denoted by I^1_t, I^2_t and I^3_t. Over the three time intervals, the maximum number of processes assigned to each machine is at most $2C_t + 1$.

Assume the algorithm has completed the schedule of superstep $t-1$ by time x_t. The schedule of superstep t will therefore be started at time x_t by migrating at most $2C_t + 1$ processes to each processor, with a cost bounded by $3T_t$. The computation is then performed in a time interval of length at most $3cT_t$, followed by a synchronization phase of cost L. Altogether, we have a cost $3(c + 1)T_t + L$ for each superstep.

This compares with a lower bound of $T_t/2 + L$ on the optimum for which we obtain a constant competitive ratio.

We are still left to describe how processes are assigned to intervals in an on-line fashion, since the length of an interval is not known till the interval is actually ended.

Once Q_i is available for processing our application, we must decide which of the processes to move to execution. This choice must be taken without knowledge of the actual duration of the interval. We propose a simple strategy that requires a limited amount of resource augmentation to be implemented. In particular, we assume that the processors of the on-line have double speed with respect to the off-line.

Let $k_1, \ldots, k_l$ be the set of different process lengths, ordered by increasing length, of processes still to be assigned to Q_i. The on-line schedule starts assigning a job of minimum length 2^{k_1} when some idle time is available. If the interval ends before the processing is completed, i.e. 2^{k_1} time units, the job will just be re-started at some later occasion. If a process of length 2^{k_j} is successfully completed, a job of length $2^{k_{(j+1) \bmod l}}$ is moved to execution.

LEMMA 5. *The on-line scheduling strategy completes all processes assigned to Q_i in interval I'.*

PROOF. We give a brief sketch of the proof. The schedule is feasible for an off-line adversary that knows in advance the duration of the intervals. Assume the off-line schedule assigns a process of size 2^{k_i} to a specific interval. The on-line algorithm will assign at most one process for every size 2^{k_j}, $j \leq i$. Since the on-line algorithm has machines of double speed, their total size sum up to at most 2^{k_i}. □

This implies the following theorem:

THEOREM 6. *Consider a program so that the off-line algorithm needs at least D steps for each of at least a constant fraction of its supersteps. Then the on-line scheduling strategy is $O(1)$ competitive in the case of strong restriction and non-uniform processes if it is equipped with machines of double speed.*

It is rather simple to observe that the assumption of processes and intervals of length equal to a power of 2 can be removed by equipping the algorithm with processors that are 4 times faster than the off-line adversary.

3. THE OFF-LINE SCHEDULING PROBLEM

The procedure outlined in this section is used as a basic subroutine to decide whether it is possible to schedule in a fractional way the processes of superstep t in a time interval $I_t = [s_t, s_t + T_t)$ of length T_t with maximum load $C_t = T_t/g$. In case of positive answer, the procedure returns an integral solution that uses at most three time intervals identical to I_t, with maximum load bounded by $2C_t + 1$.

We recall that we restrict the length of the processes and the length of the intervals to be a power of 2. Moreover, variables $l_{i,k}$ denote the number of processes of length 2^k assigned to processor Q_i. Let $b_{i,k}$ be the total size of intervals of length at least 2^k available on machine Q_i. Let m_k be the total number of processes of length 2^k. We formulate the problem of scheduling a superstep into interval I_t as the following integer linear program:

$$\begin{aligned} \sum_{j \geq k} 2^j l_{i,j} &\leq b_{i,k}, \forall i, k \\ \sum_i l_{i,k} &\geq m_k, \forall k \\ \sum_k l_{i,k} &\leq C_t, \forall i \\ l_{i,k} &\in N_0, \forall i, k \end{aligned}$$

There exists a simple algorithm that transforms an integral solution to the linear program into a feasible schedule. The algorithm simply assigns processes to intervals in decreasing order of length.

LEMMA 7. *A feasible integral solution to the linear program can be converted into a feasible schedule.*

PROOF. We prove that at any stage of the algorithm that assigns processes to intervals, the following two properties hold:

1. $$\sum_{j \geq k} 2^j l_{i,j} \leq b_{i,k}, \forall i, k,$$

   where $l_{i,j}$ denotes the number of jobs of size 2^j still left to be assigned to Q_i, $b_{i,k}$ is the total size of intervals of length at least 2^k still available on Q_i.

2. If $l_{i,j} > 0$, then 2^j time units are still available into an interval on Q_i.

The proof is by induction. The two conditions clearly hold for the feasible integral solution. At the generic stage of the algorithm, the process of largest length, say 2^j, is assigned to an interval of size at least 2^j, say on machine Q_i.

After the assignment, both sides of the equations indexed by i and $k \leq j$, are decreased by 2^j. The equations will then hold after the assignment. Moreover, given the fact that jobs are assigned by decreasing size, and all processes and interval lengths are powers of 2, the interval that receives the assignment of a process is still left with a number of time units that is a multiple integer of 2^j (including 0.) □

In the following we give a NP-completeness result even in the case in which the lengths of the processes and the size of the intervals are powers of 2. The proof of the following theorem is given in Appendix.

THEOREM 8. *It is NP-complete to decide whether the linear program has a feasible integral solution.*

It is also rather easy to see that a simple greedy heuristic solves the problem in polynomial time if the constraint on congestion is relaxed.

We then relax the integrality constraints on variables $l_{i,k}$ and check for feasibility of the resulting fractional linear program. If a feasible solution exists, this is transformed into a schedule that uses at most three time intervals identical to I_t, denoted by I_t^1, I_t^2 and I_t^3, with load at most $2C_t + 1$. We also denote by $l^*_{i,k}$ a fractional feasible optimal solution for the linear program, and by $l'_{i,k}$ the integral solution we construct.

The first step of the schedule consists in assigning $\lfloor l^*_{i,k} \rfloor$ processes of size 2^k to processor Q_i. Denote $\overline{l}^*_{i,k} = l^*_{i,k} - \lfloor l^*_{i,k} \rfloor$. We are still left to assign $\sum_i \overline{l}^*_{i,k}$ processes of size 2^k.

In the following we illustrate an algorithm that turns the fractional assignment $\overline{l}^*_{i,k}$ into a new fractional assignment $\overline{l}_{i,k}$ holding a set of properties that allow to produce a feasible integral schedule with low congestion.

The assignment $\overline{l}_{i,k}$ holds for every machine Q_i the following 3 properties:

1. $\sum_k \overline{l}_{i,k} \leq \sum_k \overline{l}^*_{i,k} + 1/2$;
2. $\overline{l}_{i,k} \in [0, 1]$; moreover $\overline{l}_{i,k} > 0$ only if $\overline{l}^*_{i,k} > 0$;
3. $\overline{l}_{i,k} \in (0, 1/2)$ implies for all processors $i' \neq i$, either $\overline{l}_{i',k} \geq 1/2$ or $\overline{l}_{i',k} = 0$,

and for every job size k the following fourth property:

4. $\sum_i \overline{l}_{i,k} = \sum_i \overline{l}^*_{i,k}$.

Define function round(x), $x \in [0, 1]$ as follows: round$(x) = 0$ if $x \in [0, 1/2)$, round$(x) = 1$ if $x \in [1/2, 1]$. The final integral solution will be:

$$l'_{i,k} = \lfloor l^*_{i,k} \rfloor + \text{round}(\overline{l}_{i,k}), \forall i, \forall k \quad (1)$$

The integral solution transforms into a scheduling spanning over the three time intervals I_t^1, I_t^2 and I_t^3. For every machine Q_i:

- Assign $\lfloor l^*_{i,k} \rfloor$ processes of size k in interval I_t^1.
- Assign the process with largest k, such that $\overline{l}_{i,k} \geq 1/2$, in interval I_t^2.
- Assign all other processes with $\overline{l}_{i,k} \geq 1/2$ in interval I_t^3.

LEMMA 9. *The schedule obtained for superstep t is feasible.*

PROOF. We first prove that all processes are assigned to suitable intervals. For every size k, we have:

$$\begin{aligned}\sum_i l'_{i,k} &= \sum_i \lfloor l^*_{i,k} \rfloor + \sum_i \text{round}(\overline{l}_{i,k}) \\ &\geq \sum_i \lfloor l^*_{i,k} \rfloor + \sum_i \overline{l}^*_{i,k} \\ &\geq m_k\end{aligned}$$

where the first inequality follows from the integrality of $\sum_i \overline{l}^*_{i,k}$ and from property 3 that states the existence of at most one machine Q_i with $\overline{l}_{i,k} \in (0, 1/2)$.

The schedule of interval I^1_t is clearly feasible since it is obtained by rounding down every $l^*_{i,k}$ to the nearest integral value. From $\overline{l}_{i,k} \leq 1$, we obtain that at most one process of each size k is to be assigned in I^2_t and I^3_t. Consider the largest k, say $\overline{k}$, with $\overline{l}_{i,k} \geq 1/2$. Property 2 ensures $\overline{l}^*_{i,k} > 0$ and therefore the existence of at least one interval of size at least 2^k on machine Q_i. The schedule of I^2_t is then feasible.

The same interval of size at least $\overline{k}$ can also host all other processes with $\overline{l}_{i,k} \geq 1/2$ in time interval I^3_t, since their sizes sum up to at most $2^{\overline{k}}$. □

LEMMA 10. *The congestion is bounded by* $2C_t + 1$.

PROOF. It follows from property 1, for every machine Q_i,

$$\begin{aligned}\sum_k l'_{i,k} &= \sum_k \lfloor l^*_{i,k} \rfloor + \sum_k \text{round}(\overline{l}_{i,k}) \\ &\leq \sum_k \lfloor l^*_{i,k} \rfloor + 2\sum_k \overline{l}^*_{i,k} + 1 \\ &\leq 2\sum_k l^*_{i,k} + 1 \\ &\leq 2C_t + 1\end{aligned}$$

□

We will now present the algorithm that transforms with a sequence of greedy moves the assignment $\overline{l}^*_{i,k}$ into an assignment $\overline{l}_{i,k}$ that holds properties 1-4. We initially set $\overline{l}_{i,k} = \overline{l}^*_{i,k}$.

The first set of greedy moves are *Cycle local improvements*, defined by:

- A set of q machines $Q_{i_1}, ..., Q_{i_q}$.
- A set of q process sizes $j_1, ..., j_q$, $j_1 = j_q$, $j_1 \neq j_2, ..., \neq j_{q-1}$, such that $\overline{l}_{i_s,j_s} \in (0, 1/2)$ and $\overline{l}_{i_{(s+1) \bmod q},j_s} \in (0, 1/2)$, $\forall s = 1, ..., q$.

Cycle local improvements uniformly increase all l_{i_s,j_s} and decrease all $l_{i_{(s+1) \bmod q},j_s}$ until a value becomes 0 or 1/2. This is a clear progress in the direction of properties 1-4 since we have one less variable $\overline{l}_{i,j} \in (0, 1/2)$. We proceed till no cycle local improvements is possible.

LEMMA 11. *Properties 2 and 4 hold after a cycle local improvement. Moreover, Property 1 holds with* $\sum_k \overline{l}_{i,k} = \sum_k \overline{l}^*_{i,k}$.

PROOF. Proof by induction. Properties 2 and 4 and Property 1 with equality clearly hold for $\overline{l}^*_{i,k}$. Assume they hold before a cycle local improvement. We show they also hold after the improvement.

For every machine Q_i, it holds $\sum_k \overline{l}_{i,k} = \sum_k \overline{l}^*_{i,k}$ after the improvement, since either Q_i is not involved in the improvement, or the values of two variables $l^*_{i,k}$ and $l^*_{i,k'}$ are increased and decreased by the same amount. Property 2 also holds since a cycle improvement involves only variables $\overline{l}_{i,k} \in (0, 1/2)$. Therefore, no variable of initial value 0 will be made positive. Moreover, the final value of a variable $\overline{l}_{i,k}$ will be at most equal to 1. Finally, property 4 is also satisfied since cycle local improvements do not alter $\sum_i \overline{l}_{i,k}$. □

When cycle local improvements are no more possible, (this will eventually happen since every cycle local improvement reduces the number of variables $\overline{l}_{i,k} \in (0, 1/2)$), we move to a second type of local improvements, *path local improvements*. A path local improvement is defined as:

- A sequence of machines $Q_{i_1}, ..., Q_{i_q}$;
- A sequence of job sizes $j_1, ..., j_q$, $j_1 \neq ... \neq j_q$ with $\overline{l}_{i_s,j_s} \in (0, 1/2)$ and $\overline{l}_{i_{s+1},j_s} \in (0, 1/2)$, $\forall s = 1, ..., q-1$.

In particular we look for *maximal path local improvements*, i.e. path local improvements whose length q cannot be extended further. We refer to Q_{s_q} as the *last machine* of the path local improvement.

We perturb the fractional solution by uniformly decreasing all l_{i_s,j_s} and increasing all l_{i_{s+1},j_s} till a variable becomes equal to 0 or to 1/2. Observe that if no cycle local improvement is possible at some stage of the algorithm, this will also be true after any sequence of path local improvements is performed.

CLAIM 12. *If machine* Q_{s_q} *is the last machine of a path local improvement, and* $\overline{l}_{i_q,j_q} = 1/2$ *after the path local improvement, then machine* Q_{s_q} *won't be part of any later path local improvement*

PROOF. By the maximality of the path local improvement, there is no size index $j \neq j_q$ and machine Q_i, $i \neq i_q$, such that $\overline{l}_{i_q,j} \in (0, 1/2)$ and $\overline{l}_{i,j} \in (0, 1/2)$. It therefore follows that Q_{s_q} won't be part of any later path local improvement. □

LEMMA 13. *After every path local improvement: i) Properties 2 and 4 hold; ii) Property 1 holds with* $\sum_k \overline{l}_{i_s,k} \leq \sum_k \overline{l}^*_{i,k}$ *for* $s = 1, ..., q-1$. *For machine* Q_{i_q}*: iii)* $\sum_k \overline{l}_{i_q,k} \leq \sum_k \overline{l}^*_{i_q,k} + 1/2$*; iv) If* $\overline{l}_{i_q,j} = 1/2$ *then Property 3 holds.*

PROOF. Proof is by induction. For the basis of the induction, Property 1 with equality, and properties 2 and 4, hold after the last cycle improvement and for the initial assignment $\overline{l}^*_{i,k}$. Properties 2 and 4 still hold after every path local improvement, since no $\overline{l}_{i,k}$ is assigned to a positive value if $\overline{l}^*_{i,k} = 0$, and the improvement does not alter $\sum_i \overline{l}_{i,k}$.

For property 1, we observe that for every machine Q_{i_s}, $s = 1, ..., q-1$, it holds $\sum_k \overline{l}_{i_s,k} \leq \sum_k \overline{l}^*_{i_s,k}$ after the improvement, since the values of two variables $\overline{l}^*_{i_s,k}$ and $l^*_{i_s,k'}$ are increased and decreased by the same amount, or, for machine Q_{i_1}, the value of a variable $\overline{l}_{i_1,k}$ is decreased.

Let us prove the claim for machine Q_{i_q}. The value of a variable $\overline{l}_{i_q,k}$ is increased by at most 1/2. Moreover, $\sum_k \overline{l}_{i_q,k} = \sum_k \overline{l}^*_{i,k} + 1/2$ only if $\overline{l}_{i_q,j_q} = 1/2$. In this case, by Claim 12, machine Q_{i_q} won't be the extreme variable of a later path local improvement. Therefore, $\sum_k \overline{l}_{i_q,k} \leq \sum_k \overline{l}^*_{i,k} + 1/2$ at the end of the algorithm. □

It is therefore clear then when no path local improvement is possible, we have converged to a solution holding properties 1-4.

The procedure above greatly simplifies if processes have uniform length. In this case it is sufficient to assign processes to processors while not exceeding the maximum load $C_t = T_t/g$ on every machine.

4. REFERENCES

[1] Albers and Schmidt. Scheduling with unexpected machine breakdowns. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 110, 2001.

[2] Baruch Awerbuch, Yossi Azar, Stefano Leonardi, and Oded Regev. Minimizing the flow time without migration. *SIAM Journal on Computing*, 31(5):1370–1382, October 2002.

[3] A. Barak and O. La'adan. The mosix multicomputer operating system for high performance cluster computing. *Journal of Future Generation Computer Systems*, 13(4–5):361–372, March 1998.

[4] Luca Becchetti, Stefano Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 548–557, N.Y., January 9–11 2000. ACM Press.

[5] Olaf Bonorden, Joachim Gehweiler, Pawel Olszta, and Rolf Wanka. PUB-Library - User Guide and Function Reference. Available at http://www.upb.de/∼pub, October 2002.

[6] Olaf Bonorden, Ben Juurlink, Ingo von Otte, and Ingo Rieping. The paderborn university bsp (pub) library. *Parallel Computing*, 13(2):187–207, February 2003.

[7] William R. Dieter and James E. Lumpp, Jr. User-level checkpointing for linuxthreads programs. In *Proceedings of the 2001 USENIX Technical Conference*, http://www.engr.uky.edu/ dieter/publications.html, June 2001.

[8] Bozhidar Dimitrov and Vernon Rego. Arachne: A portable threads system supporting migrant threads on heterogeneous network farms. *IEEE Transactions on Parallel and Distributed Systems*, 9(5):459–469, 1998.

[9] distributed.net project homepage. http://www.distributed.net.

[10] Folding@home project homepage. http://folding.stanford.edu.

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

[12] Joseph Gil, Friedhelm Meyer auf der Heide, and Avi Wigderson. The tree model for hashing: lower and upper bounds. *SIAM Journal on Computing*, 25(5):939–955, 1996.

[13] Torben Hagerup. The log-star revolution. In *Proceedings 9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 307–316. Springer-Verlag, 1992.

[14] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance [scheduling problems]. In IEEE, editor, *36th Annual Symposium on Foundations of Computer Science: October 23–25, 1995, Milwaukee, Wisconsin*, pages 214–221, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1995. IEEE Computer Society Press.

[15] Bala Kalyanasundaram and Kirk R. Pruhs. Eliminating migration in multi-processor scheduling. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 499–506, N.Y., January 17–19 1999. ACM-SIAM.

[16] Gilad Koren, Emanuel Dar, and Amihood Amir. The power of migration in multiprocessor scheduling of real-time systems. *SIAM Journal on Computing*, 30(2):511–527, April 2001.

[17] Michael Litzkow, Todd Tannenbaum, Jim Basney, and Miron Livny. Checkpoint and migration of unix processes in the condor distributed processing system. Technical Report 1346, University of Wisconsin-Madison Computer Sciences, April 1997.

[18] Yossi Matias and Uzi Vishkin. Converting high probability into nearly constant time - with applications to parallel hashing. In *Proceedings 23rd Annual ACM Symposium on Theory of Computing*, pages 307–316, 1991.

[19] Mark Nuttall. A brief summary of systems providing process or object migration facilities. *Operating Systems Review*, 28(4):64–80, 1994.

[20] Allesandro Panconesi and Aravind Srinivasan. Fast randomized algorithms for distributed edge coloring. In *Proceedings 11th Annual ACM Symposium on Principles of Distributed Computing*, pages 251–262, 1992.

[21] Paderborn University BSP Library. http://www.upb.de/∼pub.

[22] Luis F. G. Sarmenta. An adaptive, fault-tolerant implementation of BSP for java-based volunteer computing systems. In *IPPS'99 Workshop on Java for Parallel and Distributed Computing*, volume 1586 of *Lecture Notes in Computer Science*, pages 763–780, http://bayanihancomputing.net, April 1999. Springer-Verlag.

[23] Seti@home project homepage. http://setiathome.ssl.berkeley.edu.

APPENDIX

Proof of Theorem 8: We reduce from 3-Partition [11] defined as follows:

Input: a set $S = \{s_1, s_2, \ldots, s_{3n}\}$ items, an integral bound T and an integral size for each item; item s_j has size a_j such that $T/4 << a_j << T/2$ and such that $\sum_j a_j = nT$.

Question: there exists a partition of S into n subsets $S_1, S_2, \ldots, S_n$ such that for all i, $i = 1, 2, \ldots, n$, $\sum_{s_j \in S_i} a_j = T$?

Note that each S_i must contain exactly three items of S and that, without loss of generality, we may assume that $T/4 + 1 \leq a_j \leq T/2 - 1$. Let $m = \max_j(a_j) + \alpha$, $\alpha = \lceil \log n \rceil + 1$. Since 3-Partition is strong NP-complete [11], we can assume, without loss of generality that m is polynomial in n.

Given an instance S of 3-Partition we define an instance S' of the scheduling problem with n processors and $3n(n-1) + nT$ processes. Each processor has $3n$ intervals and, for all processors, the length of the j-th interval is 2^{mj}.

For each item s_j, $s_j \in S$, there exist $(a_j + n - 1)$ processes (that we call *class* j processes) whose lengths are defined as follows:

- $n - 1$ identical processes B_j each with length 2^{mj}
- a_j processes $A_{j,k}$, $k = 1, 2, \ldots, a_j$; $A_{j,k}$, $k = 1, 2, \ldots, a_j - 1$, has length $w(A_{j,k}) = 2^{mj-k}$; A_{j,a_j} has length $w(A_{j,a_j}) = 2^{mj-a_j+1}$.

It is easy to check that, for each item s_j, $\sum_k w(A_{j,k}) = 2^{mj}$; hence the total length of all class j jobs, $j = 1, 2, \ldots, 3n$ is $n2^{mj}$. Furthermore, since the total length of all intervals is equal to the total length of all processes, there is no idle time in a feasible schedule.

Since m is polynomial in n, then the reduction is polynomial in the length of a binary encoding of S. We will now show that there exists a schedule with congestion $C = 3(n-1) + T$ if and only if instance S of 3-Partition has solution.

Given a solution $S_1, S_2, \ldots, S_n$ of instance S of 3-Partition we obtain a solution of S' as follows. The j-th interval of processor Q_i contains all $A_{j,k}$, $k = 1, 2, \ldots, a_j$, processes if $s_j \in S_i$; otherwise it contains one of the $n-1$ B_j processes.

Since S_i, $i = 1, 2, \ldots, n$, is a partition of S then $\sum_k w(A_{j,k}) = 2^{mj}$ implies that the obtained schedule is feasible.

Since S_i, $i = 1, 2, \ldots, n$, contains 3 items with total size equal to T it follows that the number of processes assigned to each processor is $3n - 3 + T$. It follows that if S has solution also S' has solution.

Now assume that S' has a feasible solution with congestion $3(n-1) + T$.

Denote by $I_{i,j}$ the j-th interval of processor i.

LEMMA 14. *Given a feasible solution S', for each i and j, the total length of class j processes assigned to the j-th interval of processor Q_i is 2^{mj}.*

PROOF. By contradiction. Let $\hat{j}$ be the highest j such that there exists a processor Q_i that does not verify the lemma. Let $z_{i,j}$ be the total processing of class j processes in the j-th interval of Q_i.

Since S' is feasible then $z_{i,\hat{j}} << 2^{m\hat{j}}$; it follows that the $\hat{j}$-th interval of Q_i schedules neither a $B_{\hat{j}}$ process nor a class j process, $j > \hat{j}$.

$z_{i,\hat{j}} << 2^{m\hat{j}}$ also implies that there exists a class $\hat{j}$ process, say $A_{\hat{j},\hat{k}}$, that is not scheduled by Q_i; it follows

$$2^{m\hat{j}} - z_{i,\hat{j}} \geq 2^{m\hat{j}-a_j+1} > 2^{m\hat{j}-m+\alpha+1}$$

In order to completely cover the $\hat{j}$-th interval of Q_i we might use class j processes, $j << \hat{j}$. Now observe that the total length of class j processes, $j << \hat{j}$, is at most

$$n \sum_{j<<\hat{j}} 2^{mj} << 2n2^{m(\hat{j}-1)} \leq 2^{m(\hat{j}-1)+\alpha} << 2^{m\hat{j}-m+\alpha+1}$$

This implies that the j-th interval is not completely covered; this contradicts the assumption that S' is feasible. □

The lemma implies that in a feasible schedule, for each j and i, either Q_i schedules one process B_j or it schedules all $A_{j,k}$ processes, $k = 1, 2, \ldots, a_j$.

We now show that, for each processor Q_i, Q_i schedules processes $A_{j,k}$ with exactly three different values of j. We preliminary observe that, since there are a total of nC processes, in a feasible schedule the congestion of each processor is exactly C.

We distinguish two cases. If Q_i schedules $A_{j,k}$ processes with two different values of j, say $j1$ and $j2$, then, by Lemma 14, the total congestion of Q_i is $3n-2+a_{j1}+a_{j2}$; since $a_{j1} + a_{j2} \leq T - 2$ it follows that the congestion of Q_i is at most $C - 1$ obtaining a contradiction. The same holds if Q_i schedules processes with less than two different j values.

If Q_i schedules $A_{j,k}$ processes with four different values of j, say $j1$ $j2$, $j3$, $j4$ then, by Lemma 14, the total congestion of Q_i is $a_{j1} + a_{j2} + a_{j3} + a_{j4} + 3n - 4 \geq C + 1$ obtaining a contradiction. The same holds if Q_i schedules processes with more than four different j values.

We have shown that, in a feasible schedule, processor Q_i, $i = 1, 2, \ldots, n$, schedules all $A_{j,k}$ processes with exactly three different values of j, say $j1$, $j2$, $j3$; it follows that the congestion of Q_i is $3(n-1) + a_{j1} + a_{j2} + a_{j3}$. Since $C = 3(n-1)+T$ it follows that $a_{j1}+a_{j2}+a_{j3} = T$; this allows to obtain a feasible solution to S completing the proof of the theorem.