DIMACS Technical Report 2006-24
November 2006

Approximation Algorithms for a Resource Allocation Problem

by

Srikrishnan Divakaran [1,2]
Dept. of Computer Science
Hofstra University
Hempstead, New York 11549

[1]This problem originated from the discussions at the Algorithms Reading Group at State University of New York, Stony Brook

[2]The author benefited from DIMACS Workshops that were held as a part of Special Focus on Mathematics and the Foundations of Computer and Information Science

---

ABSTRACT

In this paper, we present approximation algorithms for a resource allocation problem that arises in the context of processor allocation among parallel tasks in massively parallel super computers. In this paper, we consider a processor allocation problem, where we are given a computing grid consisting of a set of processors connected in the form of an linear or circular list. When a job arrives, a set of available processors from the grid is assigned to that job. One of the objectives in allocating processors is to minimize communication overheads caused by overlapping jobs. That is, assign processors to minimize communication overheads caused by non-contiguous allocation of processors.

The processor problem is a variant of the two-dimensional packing problem, where objects have placement constraints and are allowed to be split into smaller objects. This problem also has applications in the design of algorithms for bandwidth allocation in computer networks and memory management in computer systems.

For the $MaxStretch$ and $TotalStretch$ metrics, the offline version of processor allocation problem is known to be strongly NP Hard. For the online version, it is known that the best achievable competitive ratio for even a randomized algorithm is $\Omega(n)$.

In this paper, we analyze the processor allocation problem in a relaxed framework, where our algorithm is given $O(1)$ more processors than the optimal offline algorithm. In this framework, for the MaxStretch and TotalStretch metrics, we present algorithms that obtains an optimal solution for the following two special cases:

- the ratio of maximum to the minimum number of processors required by jobs is bounded by a constant
- the ratio of maximum to the minimum processing time of jobs is bounded by a constant

Key words: Analysis of algorithms; Operations Research; Approximation algorithms; Online algorithms; Resource Allocation; Multi-dimensional Bin Packing;

1 Introduction

We consider a resource allocation problem [1,2,4,10,16,19] that arise in computation grids while allocating resources among competing multi-media applications. In multi-media applications, each task is independent and requires certain units of computing power (i.e. processors, memory and hard disk) for a certain interval of time. In a computation grid, you have an array of processors, memory and disks connected in a defined topology. When a job arrives, a set of available resources from the grid is assigned to that job. One of the objectives in allocating resources is to minimize communication overheads caused by overlapping jobs. That is, assign resources to minimize communication overheads caused by fragmented resource allocation (i.e. non-contiguous allocation of processors, memory and disk space).

The resource allocation problem in the case of a single resource is a variant of the two-dimensional packing problem[5,6,7,8,9,13,17] where each job can be considered as a rectangular object that has placement constraints and can be split into smaller rectangles. This problem has applications in the design of algorithms for bandwidth allocation[3] in computer networks and memory management[14,15] in computer systems.

This resource allocation problem in the context of processor allocation has been previously studied by Arkin et al. [1,2] and Subramani et al.[19]. In their studies, they motivate the importance of processor locality while allocating processors by showing that communication over heads caused by non-contiguous allocation of jobs that communicate heavily can affect the run time performance significantly.

For the processor allocation problem, Arkin et al. [1,2] introduced various metrics called *span metrics* for measuring processor locality. For two of these span metrics $MaxStretch$ and $TotalStretch$, Arkin et al [1,2] established the offline version of processor allocation to be *Strongly NP-Hard*. For the online version, they showed that the best acheivable competitive ratio for even a randomized algorithm is $\Omega(n)$.

In this paper, we analyze the processor problem in a relaxed framework[12], where our algorithm is given $O(1)$ more processors than the optimal offline algorithm. In this framework, for the MaxStretch and TotalStretch metrics, we present algorithms that obtains an optimal solution for the following two special cases:

- the ratio of maximum to the minimum number of processors required by jobs is bounded by a constant
- the ratio of maximum to the minimum processing time of jobs is bounded by a constant

The remainder of this paper is organized as follows. In Section 2 we provide a summary of relevant terms and definitions that we use throughout this paper. In Section 3, we first describe our model and the processor allocation problem, then we present related work and our results. In Section 4, we summarize the known complexity and approximability results on processor allocation problem for the MaxStretch and TotalStretch metrics. In Section 5, we present our algorithms for special cases of processor allocation that we later used as building blocks for designing online and offline approximation algorithms for processor allocation. In Section 6, we present online and offline algorithms for processor allocation that obtain optimal solutions in a relaxed framework where our algorithm uses $O(1)$ more processors than the optimal offline algorithm to which it is compared. Finally, in Section 7, we present our conclusions.

2 Definitions

For convenience, we list most of the definitions that are used in the rest of the paper.

Let

- $J = (J_1, J_2, ..., J_n)$ be any sequence of n jobs
- J_i denotes the ith job in J
- a_i denotes the arrival time of J_i
- t_i denotes the processing time of J_i
- p_i denotes the number of processors required by J_i
- $J_{[i..j]}$ denote the subsequence of jobs $J_i, ..., J_j \in J$
- $t_{min}(J) = min_{J_i \in J} t_i$ denote the minimum processing time required by any job in J
- $t_{max}(J) = max_{J_i \in J} t_i$ denote the maximum processing time required by any job in J
- $p_{min}(J) = min_{J_i \in J} p_i$ denote the minimum number of processors required by any job in J
- $p_{max}(J) = max_{J_i \in J} p_i$ denote the maximum number of processors required by any job in J
- $C_t(J) = t_{max}(J)/t_{min}(J)$
- $C_p(J) = p_{max}(J)/p_{min}(J)$
- $J^j = \{J_i \in J : 2^{j-1} p_{min}(J) < t_i \leq 2^j p_{min}(J)\}$
- $S = \{P_1, P_2, ..., P_N\}$ denote a set of N contiguous processors with labels 1 to N
- $P_{[i..j]}$ denote the set of contiguous processors labelled i to j
- a *block* denote a sequence of contiguous processors that are allotted as a group
- the size of a block is the number of processors in that block
- $\Phi(J)$ denote the set of feasible processor allocations for J
- $N^\sigma(J)$ denote the maximum number of processors used in σ
- $N_{min}(J) = min_{\sigma \in \Phi(J)} N^\sigma(J)$ be the minimum number of processors required by any feasible schedule
- $Active^t(J) = \{J_i \in J : t \in [a_i, a_i + t_i]\}$ denote the set of jobs that are active at time t
- OPT stands for any optimal offline algorithm for processor allocation
- $N^{OPT}(J)$ - the number of processors used by OPT for serving jobs in J
- $N^A(J)$ - the number of processors used by algorithm A for serving jobs in J

3 Problem Description

We are given a computer system consisting of an array of N identical processors labelled 1 to N such that for integers i, j and $i > j$, processor i and processor j are at a distance $j - i$ apart. This system services a sequence $J = (J_1, J_2, ..., J_m)$ of jobs that arrive over a period of time. For $i \in [1..m]$, job J_i has an arrival time a_i, a requested number of processors p_i, and a processing time t_i. The jobs in J arrive *online* and are *temporary*. That is the job J_i is only known to the system at time a_i and *the processing time t_i is known only after the job J_i completes.* In our model, a job J_i on arrival at time a_i, is assigned p_i processors (not necessarily contiguous) for t_i time units. Once the p_i processors are assigned to J_i, the job cannot be migrated to any other processor nor can the allotted processors be reassigned to any other job until that job completes.

For a given job sequence J, an assignment σ determines for each job $J_i \in J$, the set of processors $P^\sigma(J_i)$ that are assigned to J_i during the time interval $[a_i, a_i + t_i]$. Let $\Phi(J)$ denote the set of all feasible assignments for J. For a given assignment σ, we define

$Stretch^\sigma(J_i)$ to be the maximum distance between any two processors assigned to J_i;

$MaxStretch^\sigma = max_{i \in [1..n]} Stretch^\sigma(J_i)$

$TotalStretch^\sigma = \sum_{i \in [1..n]} Stretch^\sigma(J_i)$

For the job sequence J, let $TotalStretch^{OPT}(MaxStretch^{OPT})$ be the optimal Total Stretch and Max Stretch respectively.

For an optimization problem P with an objective function f, real constants c and d, an algorithm A is said to be a c-approximation algorithm, if for any sequence J of jobs, A determines an allocation σ whose objective function value is within c times the optimal solution. That is, $f^\sigma(J) \leq c * f^{OPT}(J)$. In addition, if at any time t, if algorithm A determines its allocation based on jobs that have arrived prior to t, then A is said to be c-competitive.

For an optimization problem P with an objective function f, for some positive real numbers $s \geq 1$ and $c \geq 1$, an algorithm A is said to be a s-speed-c-approximation algorithm, if for any sequence J of jobs, A uses at most s times more processors that any optimal algorithm and determines a solution that is within c times the optimal solution. In addition, if at any time t, if algorithm A determines its allocation based on jobs that have arrived prior to t, then A is said to be S-speed-c-competitive.

Now, we will present the definition of the following processor allocation problems that we consider in this paper.

TotalStretch Problem: Given a system consisting of an array of N identical processors and an arbitrary job sequence J, determine a schedule $\sigma \in \Phi(J)$ that minimizes $TotalStretch^\sigma$.

MaxStretch Problem: Given a system consisting of an array of N identical processors and an arbitrary job sequence J, determine a schedule $\sigma \in \Phi(J)$ that minimizes $MaxStretch^{\sigma}$.

Note: For any job sequence J and any feasible assignment σ, $MaxStretch^{\sigma} \geq max_{i\in[1..n]}p_i$ $TotalStretch^{\sigma} \geq \sum_{i\in[1..n]} p_i$.

3.1 Related Problems

The processor allocation problem can be considered as a fractional two dimensional bin packing problem with placement constraints and splitting costs. More precisely, given a computation grid consisting of N processors and a job sequence $J = (J_1, ..., J_n)$, we can view the processor allocation problem as follows:

> The computational grid can be considered as a rectangle with width N and height $max_{i\in[1..n]}(a_i + t_i)$. For $J_i \in J$, each job $J_i \in J$ can be viewed as a rectangle of width p_i and height t_i. In addition, each rectangle has placement constraints (i.e. can only be placed within the height interval $[a_i, a_i + t_i]$). However, we are free to split each rectangle vertically into any number of rectangles of unit width. Depending on the performance metric, there will be a cost associated with splitting the rectangles. Our objective is to find a packing that satisfies the placement constraints and minimizes the splitting costs.

The processor allocation problem has been widely studied by Arkin et al[1,2], and Subramani et al[19] in the context of resource allocation strategies to optmize processor locality. Arkin et al[1,2] present both theoretical and experimental results of their study of processor allocation in CPlant, a multi-user massively parallel computer at Sandia National Labs. They proposed performance metrics called *span metrics* for measuring processor locality. For two of these span metrics MaxStretch and TotalStretch, Arkin et al [1,2] established the offline version of processor allocation to be Strongly NP-Hard. For the online version, they showed that the best acheivable competitive ratio for even a randomized algorithm is $\Omega(n)$.

In addition, for these span metrics they reduced the multi-dimensional allocation problem into one dimensional allocation problem and presented various experimental results on one dimensional allocation strategies like first fit, best fit and sum of squares. However they do not provide any performance guarantees for their allocation strategies.

Subramani et al[19] studied resource allocation algorithms in the CPlant system. Their strategy for processor allocation was motivated by buddy strategy for memory allocation. However, in their model jobs can be held back (i.e. delayed) where in our model the jobs have to assigned processors immediately. More recently, Divakaran[10] studied the processor allocation problem in a relaxed framework where the algorithm is given $O(1)$ times more resources than the optimal algorithm to which it is compared. In this relaxed framework they obtain optimal solutions for some restricted versions of the TotalStretch problem.

The processor allocation problem is also related to problems that have been mainly studied in the context of online bin packing[3,7,8,9], multi-dimensional bin packing[5,6,9,13,17] and memory management[14,15]. The offline version of the one dimensional bin packing problem is NP-hard[11]. However, there are a wide range of results on approximation algorithms for the online and offline version of one dimensional bin packing and we refer the reader to the results in [7,9]. The multi-dimensional bin packing results has also been widely studied and we refer the reader to the results in [5,8,9,16]. However, due to the placement constraints and splitting costs, the existing online and two dimensional packing results do not apply to the processor allocation problem. Another related problem as described by Arkin et al. in [1,2] is the memory allocation problem. In this problem there is an array of memory, and contiguous sub-arrays are allocated or deallocated online. One objective is to minimize the highest memory address used and consequently the required memory size. Memory allocation differs from processor allocation because memory allocators leave empty space to guarantee contiguity and are allowed to refuse requests that do not fit contiguously.

3.2 Our Results

In this paper, we analyze the processor problem in a relaxed framework, where our algorithm is given fractionally more processors than the optimal offline algorithm. In this framework, for the MaxStretch and TotalStretch metrics, we present algorithms that obtains an optimal solution for the following two special cases:

- the ratio of maximum to the minimum number of processors required by jobs is bounded by a constant

- the ratio of maximum to the minimum processing time of jobs is bounded by a constant

These results also apply to two-dimensional and three dimensional grids.

4 Problem Complexity

In this section for the MaxStretch and TotalStretch problems, we summarize the known complexity and lower bound results on online algorithms.

Theorem 1 *The MaxStretch and TotalStretch problems are strongly NP-hard.*

This result was established by Arkin et al[1,2]. They reduced the numerical $3-DimensionalMatching$ problem both to the MaxStretch as well as the TotalStretch problems. We refer the reader to the results in [1,2].

Theorem 2 *The MaximumStretch and TotalStretch problems cannot be approximated better than $\Omega(n)$ even by a randomized algorithm.*

This result was also established by Arkin et al[1,2]. We refer the reader to the results in [1, 2].

5 Building Blocks

In this section, we consider algorithms for processor allocation in a relaxed framework where our algorithm is given more processors than the optimal offline algorithm to which it is compared. In this relaxed framework, we present algorithms that obtain optimal solutions for the MaxStretch and TotalStretch metrics for the following two restricted versions of processor allocation:

(1) the ratio of maximum to the minimum processing times of jobs is at most two

(2) the ratio of the maximum to the minimum number of processors required by jobs is at most two

In the subsequent sections, we use the algorithms defined in this section as building blocks to design online and offline approximation algorithms for more general processor allocation problems. The above two restricted versions of the processor allocation problem have many similarities. However, there are some differences that result in different approximability results for these two problems. In addition, in our model, at the time of job arrival the number of processors required by that job is known whereas the processing time of that job is not known. So this has different implications on extending the offline results to the online scenario. The rest of this section is organized as follows. In Section 5.1, we present processor allocation algorithms for circular lists when the ratio of maximum to minimum processing times of jobs is at most two. Then we extend this algorithm for linear lists. In Section 5.2, we present processor allocation algorithms for circular and linear lists when the ratio of maximum to minimum number of processors required by jobs is at most two. In Section 5.3, we present the analysis of the algorithms presented in Sections 5.1 and 5.2.

5.1 Processor Allocation when the ratio of the maximum to the minimum processing time of jobs is at most two

Processor Allocation in Circular Lists: Now, we present an algorithm for processor allocation in circular lists under the following assumptions:

- $t_{max}(J)/t_{min}(J) \leq 2$
- $N \geq 2N_{min}(J)$

Key Idea: If all jobs complete in order of their arrival times, then by assigning processors to jobs in a round robin fashion we can ensure that at all times all the available(free) processors are contiguous.

Circular-Allocator(J, N, S)
Inputs:
J : the Job Sequence with $t_{max}(J)/t_{min}(J) \leq 2$
N : the number of processors available to the algorithm
S : a set of N processors connected in the form of a circular list
begin
$start = 1$
For $J_i \in J$
begin
If the p_i processors $P_{[start..(start+p_i-1)\%N]}$ are free
then
Let $j = min\{2^j : 2^{j-1} < t_i \leq 2^j\}$
assign p_i processors $P_{[start..(start+p_i-1)\%N]}$ to J_i for 2^j
time units during the time interval $[a_i, a_i + 2^j]$
$start = (start + p_i)\%N$
else
return due to insufficient number of processors
end
end

Processor Allocation in Linear Lists: Now, we present an algorithm for processor allocation in linear lists under the following assumptions:

- $t_{max}(J)/t_{min}(J) \leq 2$
- $N \geq 2N_{min}(J) + p_{max}(J)$

Key Idea: The algorithm assigns processors in a round robin fashion similar to the algorithm for circular lists except that in the case of linear list processors assigned to a job can be split (i.e. non-contiguous) when we wrap around on reaching the right end of the list. This split processor assignment is avoided by using $p_{max}(J)$ additional processors that are exclusively assigned to jobs that would have been split in a round robin assignment.

Linear-Allocator(J, N, S)
Inputs:
J : the Job Sequence with $t_{max}(J)/t_{min}(J) \leq 2$
N : the number of processors available to the algorithm
S : a set of N processors connected in the form of a linear list
begin
Let $start = 1$, $N_{Split} = p_{max}(J)$, $N_{contiguous} = 2N_{min}(J)$
Let $S_{contiguous} = \{P_1, ...P_{N_{contiguous}}\}$ be the first $N_{circular}$ processors in S
Let $S_{split} = \{P_{N_{contiguous}+1} ...P_N\}$ be the last N_{split} processors in S
For $J_i \in J$
begin

if $start + p_i > N_{contiguous}$
then
if the first p_i processors in S_{split} are available
then
assign the first p_i processors in S_{split} to J_i for t_i
time units during the time interval $[a_i, a_i + t_i]$
$start = (start + p_i)\% N_{contiguous}$
else
return due to insufficient number of processors
else
if the p_i processors $P_{[start,start+p_i-1]}$ are available
then
Let $j = min\{2^j : 2^{j-1} < t_i \leq 2^j\}$
assign p_i processors $P_{[start..(start+p_i-1)\%N_{contiguous}]}$ to J_i for
2^j time units during the time interval $[a_i, a_i + 2^j]$
$start = (start + p_i)\% N_{contiguous}$
else
return due to insufficient number of processors
end
end

5.2 Processor Allocation when the ratio of the maximum to minimum number of processors used by jobs is at most two

We now present an algorithm for processor allocation in circular and linear lists under the following assumptions:

- $p_{max}(J)/p_{min}(J) \leq 2$
- $N \geq 2N_{min}(J)$

Key Idea: We will partition the N processors into *blocks* of size $2p_{min}(J)$, where a *block* is a set of contiguous processors that are assigned and freed as a group. Then, for each job $J_i \in J$, we will assign any available (free) block.

Algorithm $Online - Allocator(J, N, S)$
Inputs:
J : the Job Sequence with $p_{max}(J)/p_{min}(J) \leq 2$
N : the number of processors that are available to the algorithm
S : a set of N processors that have been split into blocks
begin
For $J_i \in J$
begin

Let $N_{adj} = min\{2^j : 2^{j-1} < p_i \leq 2^j\}$
If there is a free block of processors of size N_{adj} in S
then
assign any free block of processors of size N_{adj} in S to job J_i until it completes
else
return due to non-availability of free block of size N_{adj}
end
end

5.3 Analysis of Building Blocks

In this subsection, we will first analyze processor allocation algorithms in circular and linear lists that obtain optimal solutions when the ratio of processing times of any pair of jobs is less than or equal to two. Then, we will analyze processor allocation algorithms in circular and linear lists that obtain optimal solutions when the ratio of the number of processors required by any pair of jobs is less than or equal to two.

5.3.1 Processor Allocation when the ratio of maximum to minimum processing times of jobs is at most two

Lemma 3 *Given a job sequence J where all jobs have processing time c and a set S consisting of a circular list of $N = N_{min}(J)$ processors, the algorithm $Circular-Allocator(J, N, S)$ determines an online assignment with $MaxStretch = p_{max}(J)$ and $TotalStretch = \sum_{J_i \in J} p_i$.*

Proof Let $J = (J_1, ..., J_n)$ be a sequence of n jobs such that for $J_i \in J$, job J_i arrives at time a_i and requires p_i processors during the time interval $[a_i, a_i + c]$. Without loss of generality, we assume that the jobs in J are arranged in non-decreasing order of their arrival times. At any time t, let $Active^t(J)$ denote the jobs in J that have arrived but not completed by time t. From the definition of $N_{min}(J)$, we can observe that

$$N_{min}(J) \geq \sum_{J_i \in Active^t(J)} p_i \quad \forall t > 0 \tag{1}$$

From the definition of Algorithm $Circular - Allocator$, we can see that either one of two following situations can happen.

(1) for each job $J_i \in J$, the algorithm assigns p_i contiguous processors during the time interval $[a_i, a_i + c]$. In this case we are done.

(2) there exists a job $J_l \in J$ for which the algorithm is unable to find p_l contiguous processors. In this case the algorithm fails.

Now, we will show that if $Circular - Allocator$ is given $N_{min}(J)$ processors then Situation (2) will not happen (i.e. the algorithm will not fail) and hence we are done. We prove this by contradiction. Let us assume that the algorithm $Circular - Allocator$ fails. Let

$J_l \in J$ be the first job for which the $Circular - Allocator$ fails. Notice that the algorithm allocates processors in a round robin manner. More formally, for any pair of consecutive jobs $J_i, J_{i+1} \in J$ the processor with the highest label assigned to job J_i will be adjacent to the processor with the lowest label assigned to job J_{i+1}. Also, the processing time of all jobs are the same, so all jobs complete in the order in which they arrive. In addition, the processors in S are connected in the form of a circular list. From the above three observations, we can see that at all times $t \geq 0$, the processors that are assigned to jobs in $Active^t(J)$ are contiguous. This also implies that that at all times the processors that are free are also contiguous. So if $Circular - Allocator$ fails to find p_l contiguous processors for job J_l then at time a_l $\sum_{J_i \in Active^{a_l}(J)} p_i > N_{min}(J)$. However, this contradicts Equation (1) and leads to a contradiction. Hence, we are done. $\square$

Lemma 4 *Given any job sequence J such that $t_{max}(J)/t_{min}(J) \leq 2$ and a set S consisting of a circular list of $N = 2N_{min}(J)$ processors then the algorithm $Circular - Allocator(J, N, S)$ determines an assignment with $MaxStretch = p_{max}(J)$ and $TotalStretch = \sum_{J_i \in J} p_i$.*

Proof Let $J = (J_1, ..., J_n)$ be a sequence of n jobs such that for $J_i \in J$, job J_i arrives at time a_i and requires p_i processors during the time interval $[a_i, a_i + t_i]$. Let $S = \{P_1, ..., P_N\}$ be a set consisting of a circular list of $N = 2N_{min}(J)$ processors. Since $t_{max}(J)/t_{min}(J) \leq 2$, there exists a unique integer l such that for all $J_i \in J$, $2^{l-1} < t_i \leq 2^l$. Notice that the algorithm $Circular - Allocator(J, N, S)$ makes the processing time of all jobs in J to be the 2^l. Let $\bar{J}$ denote the transformed job sequence where for $\bar{J}_i \in \bar{J}$, the arrival time $\bar{a}_i = a_i$, requires $\bar{p}_i = p_i$ processors for $\bar{t}_i = 2^l$ time units. Notice in $\bar{J}$ all jobs have the same processing time. Now, from Lemma 3, we know that $Circular - Allocator(\bar{J}, N_{min}(\bar{J}), S)$ obtains a schedule $\bar{\sigma}$ such that $MaxStretch^{\bar{\sigma}} = p_{max}(J)$ and $TotalStretch^{\bar{\sigma}} = \sum_{J_i \in J} p_i$. Also, in $\bar{J}$ the processing time of each job J_i is at most doubled. So $N_{min}(\bar{J}) \leq 2N_{min}(J)$. Therefore, $Circular - Allocator$ using at most $2N_{min}(J)$ processors obtains an schedule such that $MaxStretch^{\bar{\sigma}} = p_{max}(J)$ and $TotalStretch^{\bar{\sigma}} = \sum_{J_i \in J} p_i$. $\square$

Lemma 5 *Let J be any job sequence such that $t_{max}(J)/t_{min}(J) \leq 2$, S_C be a circular list of $N_C = 2N_{min}(J)$ processors, S_L be a linear list of $N_L = 2N_{min}(J) + p_{max}(J)$ processors. If $Circular - Allocator(J, S_C, N_C)$ determines an assignment with $MaxStretch = p_{max}(J)$ and $TotalStretch = \sum_{J_i \in J} p_i$ then $Linear - Allocator(J, S_L, N_L)$ will also determine an assignment with $MaxStretch = p_{max}(J)$ and $TotalStretch = \sum_{J_i \in J} p_i$.*

Proof Let σ be the assignment determined by $Circular - Allocator(J, N_C, S_C)$. Let $\bar{\sigma}$ be the assignment determined by $Linear - Allocator(J, N_L, S_L)$. We basically need to show that if $Circular - Allocator$ using N_C processors determines an assignment σ such that $Max - Stretch = p_{max}(J)$ and $Total - Stretch = \sum_{J_i \in J} p_i$ then $Linear - Allocator$ using N_L processors determines $\bar{\sigma}$ such that $Max - Stretch^{\bar{\sigma}} = p_{max}(J)$ and $Total - Stretch^{\bar{\sigma}} = \sum_{J_i \in J} p_i$.

From the definition of Algorithm $Linear - Allocator$, we can see that either one of two following situations can happen.

(1) for each job $J_i \in J$, the algorithm assigns immediately on arrival p_i contiguous processors for 2^j time units, where $2^{j-1} < t_i \leq 2^j$. In this case we are done.

(2) there exists a job $J_l \in J$ for which the algorithm is unable to find p_l contiguous processors. In this case the algorithm fails.

Now, we will show that if $Circular - Allocator$ using a set S_C of N_C processors does not fail then $Linear - Allocator$ using a set S_L of N_L processors will also not fail. That is situation (2) will not happen (i.e. the algorithm will not fail) and hence we will be done.

Now, we establish the lemma by partitioning the jobs in J into *split* and *contiguous* jobs and then showing that the split jobs in J require at most $2p_{max}(J)$ processors and the contiguous processors require at most $2N_{min}(J)$ processors. We define *split* and *contiguous* jobs in terms of the assignment σ determined by $Circular - Allocator$. A job $J_i \in J$ is said to be contiguous if in σ all processors assigned to J_i have consecutive labels and split otherwise. Let J_{split} be the set of jobs in J that are split and $J_{contiguous}$ be the set of jobs in J that are contiguous. Now, $Linear - Allocator$ will set aside the first $2N_{min}(J)$ processors in S_L for contiguous jobs and the remaining p_{max} processors in S_L for the split jobs. More precisely, the Linear-Allocator algorithm will allocate processors for $J_i \in J$ as follows: If J_i is a contiguous job then it will allocate the processors in S_L in exactly the same way as σ (Circular-Allocator). If J_i is a split job then it will allocate p_i processors from the $p_{max}(J)$ processors in S_L that have been set aside for split jobs.

Now, we will obtain a bound on the number of processors required for contiguous jobs. We can observe that the processor assignment for contiguous jobs is the same in σ as well as $\bar{\sigma}$. From Lemma 4, we know that the assignment σ requires at most $2N_{min}(J)$ processors. Therefore, the number of processors required by $Linear - Allocator$ for $J_{contiguous}$ is at most $2N_{min}(J)$. Now, we will obtain a bound on the number of processors required for split jobs. Now, we will show that no two split jobs overlap and hence the maximum number of processors needed for jobs in J_{split} is at most $p_{max}(J)$. Let J_k and $J_l, k < l$ be any two split jobs in J_{split}. We need to show that J_k and J_l do no overlap. Suppose the jobs J_k and J_l overlap. Then, since jobs complete in the order of their arrival times, if jobs J_k and J_l overlap then J_l will overlap with all jobs that arrived in the interval $[a_k, a_l]$. This implies that at time a_l $Active^{a_l}(J) \geq \sum_{i \in [k..l]} p_i > N_{min}(J)$. This is not possible. Hence no two split jobs overlap. Therefore, the maximum number of processors required for split jobs is at most p_{max}. Therefore, using $2N_{min}(J) + p_{max}(J)$ processors $linear - allocator$ obtains an optimal solution. □

5.3.2 Processor Allocation when the number of processors required by jobs are bounded within a factor of two

Lemma 6 *Given a job sequence J where all jobs require the same number of processors, a set S consisting of a circular or linear list of $N = N_{min}(J)$ processors, the algorithm $Online - Allocator(J, N, S)$ obtains an online assignment with $MaxStretch = p_{max}(J)$ and $TotalStretch = \sum_{J_i \in J} p_i$.*

Proof Let c be some integer constant, $J = (J_1, ..., J_n)$ be a sequence of n jobs such that for $i \in [1..n]$, job J_i arrives at time a_i and requires c processors during the time interval $[a_i, a_i + t_i]$. Since all jobs require the same number of processors, it is very easy to see that if we split the N processors into blocks of size c each then we can allocate and free processors in a block as a group and hence we can assign any free block to any job without any processor fragmentation. Hence the result. $\square$

Lemma 7 *Given any job sequence J such that $p_{max}(J)/p_{min}(J) \leq 2$ and a set S consisting of a linear or circular list of $N \geq 2N_{min}(J)$ processors then the algorithm $Online - Allocator(J, N, S)$ determines an assignment with $MaxStretch = p_{max}(J)$ and $TotalStretch = \sum_{J_i \in J} p_i$.*

Proof let $J = (J_1, ..., J_n)$ be a sequence of n jobs such that for $J_i \in J$, job J_i arrives at time a_i and requires p_i processors during the time interval $[a_i, a_i + t_i]$. Let $S = \{P_1, ..., P_N\}$ be a set consisting of a circular list of $N = 2N_{min}(J)$ processors. Since $p_{max}(J)/p_{min}(J) \leq 2$, there exists an integer l such that for all $J_i \in J$, $2^{l-1} < p_i \leq 2^l$. Notice that the algorithm $Online - Allocator(J, N, S)$ makes the number of processors required by all jobs in J to be the same by setting the the number of processors required by every job in J to 2^l. Let $\bar{J}$ denote the transformed job sequence. Now, from Lemma 6, we know that $Online - Allocator(\bar{J}, N_{min}(\bar{J}), S)$ obtains a schedule $\bar{\sigma}$ such that $MaxStretch^{\bar{\sigma}} = p_{max}(J)$ and $TotalStretch^{\bar{\sigma}} = \sum_{J_i \in J} p_i$. Notice that in $\bar{J}$ the number of processors required for each job J_i is at most doubled. So $N_{min}(\bar{J}) \leq 2N_{min}(J)$. Therefore, $Online - Allocator$ using at most $2N_{min}(J)$ processors obtains an schedule such that $MaxStretch^{\bar{\sigma}} = p_{max}(J)$ and $TotalStretch^{\bar{\sigma}} = \sum_{J_i \in J} p_i$. $\square$

6 $O(1)$-speed-1-approximation Algorithms

In this section, we present online and offline algorithms for the $MaxStretch$ and $TotalStretch$ metrics in a relaxed framework where our algorithm is given $O(1)$ times more processors that the optimal offline algorithm to which it is compared. In this relaxed framework, we use the algorithms presented in Section 5 as building blocks to design algorithms that obtain optimal solutions for the $MaxStretch$ and $TotalStretch$ metrics for the following two special cases:

(1) the ratio of maximum to the minimum processing times of jobs in J is bounded by a constant

(2) the ratio of the maximum to the minimum number of processors required by jobs in J is bounded by a constant

This section is organized as follows. In Section 6.1 we present an offline algorithm for processor allocation in linear and circular lists when the the ratio of maximum to the minimum processing times of jobs is bounded by a constant. In Section 6.2 assuming we have an upper bound on the minimum number of processors required in any feasible processor allocation, we present an online algorithm for processor allocation in linear and circular lists when the the ratio of maximum to the minimum number of processors required by jobs in J is bounded by a constant. Then using *a standard doubling technique* we extend this online algorithm to work without assuming an upper bound. In Section 6.3 we present the analysis of the offline and online algorithms presented in Sections 6.1 and 6.2.

6.1 Processor Allocation when the ratio of the maximum to minimum processing times of jobs is constant

Now, we present an offline algorithm for processor allocation in both linear and circular lists under the following assumptions:

- $t_{max}(J)/t_{min}(J) \leq c$
- $N \geq log_2(c)(2N_{min}(J) + p_{max})$

Key Idea: The algorithm basically partitions the job sequence J into $log_2(t_{max}(J)/t_{min}(J))$ subsequences such that the ratio of processing times of jobs within each subsequence is less than equal to two. The algorithm then independently assigns processors to each of these $log_2(t_{max}(J)/t_{min}(J))$ subsequences using $Linear - Allocator$ as a subroutine.

$General - Offline - Allocator(J, N, S)$
Inputs:
J : the Job Sequence with $t_{max}(J)/t_{min}(J) \leq c$
N : the number of processors that are available to the algorithm
S : a sequence of N processors that are connected in the form of a circular or linear list
begin
Let $C_t(J) = log_2(t_{max}(J)/t_{min}(J))$
Let $num = 2N_{min}(J) + p_{max}(J)$
For $j \in [1..C_t(J)]$
begin
Let $S_j = \{P_{(j-1)*num+1}, ..., P_{j*num}\}$
Let $J^j = \{J_i \in J : 2^{j-1}t_{min}(J) < t_i \leq 2^j t_{min}(J)\}$
$Linear - Processor - Allocator(J^j, num, S_j)$
end
end

Remark 1: In our model, the processing times of jobs are not known at the time of job arrival. So in our model this algorithm cannot be used to solve the online version of the problem. However, in a slightly weaker model where the processing times of jobs are made available at the time of job arrival this idea can be used to obtain an online algorithm.

Remark 2: The key idea used in this algorithm can be extended to solve the offline version of processor allocation problem when the ratio of the maximum to the minimum number of processors required by jobs is bounded by a constant. However, due to differences in allocation structure the of quality of approximation (i.e. the number of processors required) will be different for these two problems.

6.2 Processor Allocation when the ratio of the maximum to minimum number of processors used by jobs is constant

Now, we present an algorithm for processor allocation in both linear and circular lists under the following assumptions:

- $p_{max}(J)/p_{min}(J) \leq c$ for some constant c
- Λ is an upper bound on $N_{min}(J)$
- $N \geq 2c\Lambda$

Key Idea: The algorithm basically partitions the set S of N processors into c partitions each consisting of 2Λ processors. The processors in partition j, for $j \in [1..c]$, are partitioned into blocks of size $p_{min}(J)2^j$ and are assigned only to jobs $J_i \in J$ such that $p_{min}(J)2^{j-1} < p_i \leq p_{min}(J)2^j$. Notice that the ratio of the number of processors required by any pair of jobs assigned to a partition is less than 2. In effect, the jobs assigned to each partition is solved independently using $Online - Allocator$ as a subroutine.

$General - Online - Allocator(J, N, S, \Lambda)$
Inputs:
J : the Job Sequence with $t_{max}(J)/t_{min}(J) \leq c$
N : the number of processors that are available to the algorithm
S : a sequence of N processors that are connected in the form of a circular or linear list
Λ: a upper bound on $N_{min}(J)$
begin
Let $C_p(J) = log_2(p_{max}(J)/p_{min}(J))$
For $j \in [1..C_p(J)]$
begin
Let $S_j = \{P_{2(j-1)\Lambda}, ..., P_{2j\Lambda}\}$
Let $N_j = p_{min}(J) * 2^j$
Partition S^j into blocks of size N_j
end
$Online - Allocator(J, 2C_p(J)\Lambda, S)$
end

Now, using *a standard doubling technique* we now construct online algorithm H that works without assuming an upper bound on $N_{min}(J)$.

Basic Idea: Let $log_2(p_{max}(J)/p_{min}(J)) \leq c$ for some constant c. The algorithm H assigns processors to jobs in J in phases. At the start, H sets the estimate $N_{est}^{(0)}$ on the number of processors required for a feasible allocation to $min\{2^j : 2^{j-1} < p_{max} < 2^j\} \leq 2p_{max}$. and the set $J^{(0)}$ of jobs in J that have not yet been assigned processors to J. For $i \in N$, during phase i, H adds a set of $2cN_{est}^{(i-1)}$ contiguous processors to the set of processors allotted during the previous phases and executes $General - Online - Allocator$ on $\bar{J}^{(i)}$. Phase i ends when $General - Online - Allocator$ fails due to insufficient number of processors. Then H sets $N_{est}^{(i)} = 2N_{est}^{(i-1)}$ and $J^{(i+1)}$ to be the set of unassigned jobs in J at the end of phase i and starts the next phase.

Algorithm $H(J)$
begin
    Let $i = 0$, $J^{(0)} = J$, $start^{(0)} = end^{(0)} = 0$
    Let $N_{est}^{(i)} = min\{2^j : 2^{j-1} < p_{max}(J) \leq 2^j\}$
    Let $C_p(J) = log_2(p_{max}(J)/p_{min}(J))$
    while ($J^{(i)}$ is not empty)
    begin
        Let $i = i + 1$
        Let $start^{(i)} = end^{(i-1)} + 1$
        Let $end^{(i)} = start^{(i)} + N_{est}^{(i-1)} C_p(J)$
        Let $P^{(i)} = \{P_{start^{(i)}}, ..., P_{end^{(i)}}\}$ denote the set of new processors allotted
            during phase i
        Let $S^{(i)} = S^{(i-1)} \bigcup P^{(i)}$ denote the set of all processors allotted up to phase i
        For $j \in [1..C_p(J)]$
        begin
            Let $P_j^{(i)} = \{P_{start^{(i)}+2(j-1)N_{est}^{(i-1)}+1}, ..., P_{start^{(i)}+2jN_{est}^{(i-1)}}\}$
            Partition the $N_{est}^{(i-1)}$ processors in $P_j^{(i)}$ into blocks of size $2^j p_{min}(J)$
        end
        Execute $General - Online - Allocator(J^{(i)}, end^{(i)}, S^{(i)}, N_{est}^{(i-1)})$
        Let $N_{est}^{(i)} = 2N_{est}^{(i-1)}$
        Let $J^{(i)}$ denote the unassigned jobs in J at the end of phase i
    end
end

Remark 3: The above Algorithm can be modified solve the online version of the processor allocation problem when the ratio of processing times are bounded by a constant. However, the number of processors required to obtain an optimal solution would be slightly more than in this situation. (Please see Remark 2)

6.3 Analysis of Offline and Online Algorithms

Theorem 8 *Given some constant $c \geq 1$, a job sequence J such that $t_{max}(J)/t_{min}(J) \leq c$ and a set S consisting of a linear or circular list of $N = log_2(c)(2N_{min}(J) + p_{max}(J))$ processors,* *the algorithm* $General - Offline - Allocator(J, N, S)$ *is an* $O(1)$*-speed-*1 *approximation algorithm for the Max-Stretch and Total-Stretch problems.*

Proof Let $c \geq 1$ be some constant, $J = (J_1, ..., J_n)$ be a sequence of n jobs such that $t_{max}(J)/t_{min}(J) \leq c$, and N be the smallest integer greater than $log_2(c)(2N_{min}(J)+p_{max}(J))$. Let G denote the algorithm $General - Offline - Allocator$, and OPT denote any optimal offline algorithm that minimizes $TotalStretch$ or $MaxStretch$. For the job sequence J, let $S^{OPT} = (P_1, ..., P_{N^{OPT}})$ be the linear or circular list of N^{OPT} processors available to OPT, σ^* be the assignment determined by OPT, $S^G = (P_1, ..., P_{N^G})$ be the linear or circular list of N^G processors available to G and σ be the assignment determined by G.

Let $num = 2N_{min}(J) + p_{max}(J)$. For $j \in [1..log_2(c)]$, let $S_j = \{P_{(j-1)num+1}, ..., P_{j*num}\}$ be the partitioning of processors in S^G into $log_2(c)$ equal sized partitions where within each partition the processors are contiguous, and $J^j = \{J_i \in J : 2^{j-1}t_{min}(J) < t_i \leq 2^j t_{min}(J)\}$ be the partitioning of the job sequence J into $log_2(c)$ subsequences where within each subsequence the ratio of the maximum to minimum processing times of jobs is ≤ 2.

Notice that the algorithm G basically splits the sequence J into $log_2(c)$ subsequences and solves the processor allocation for each subsequence as an independent sub-problem. That is, for $j \in [1..log_2(c)]$, processors in S^j are dedicated exclusively for jobs in J^j and processor assignment for subsequence J^j is done using $linear - Allocator(J^j, S^j)$ as a sub-routine.

We will now prove this theorem by first showing that G uses only $O(1)$ times more processors than OPT. Then, we will show that G obtains an optimal solution for both $Max - Stretch$ and $Total - Stretch$ problems.

Notice that for $j \in [1..log_2(c)]$, G uses $2(N_{min}(J) + p_{max}(J)) \leq 4N_{min}(J)$ processors for each subsequence J^j. Therefore, the total number of processors that G uses is $2log_2(c)(N_{min}(J) + p_{max}(J))$. The number of processors used by OPT is $N^{OPT} \geq N_{min}(J)$. Therefore, G uses $O(log_2(c))$ times more processors than OPT. Since, c is a constant, G uses $O(1)$ more resources than OPT.

Now, we will show that G obtains an optimal solution. For $j \in [1..log_2(c)]$, let $\sigma^{(j)}$ denote the assignment obtained by $Linear - Allocator(J^{(j)}, S^{(j)})$. Notice that $\sigma = \bigcup_{j\in[1..log_2(c)]} \sigma^j$. From Lemma 3, we know that

$$Max - Stretch^{\sigma_j} = p_{max}(J^j) \leq p_{max}(J), for j \in [1..log_2(c)] \tag{2}$$

$$Total - Stretch^{\sigma_j} = \sum_{J_i \in J^f} p_i, for j \in [1..log_2(c)] \tag{3}$$

From Equation 2, taking the maximum over $j \in [1..log_2(c)]$, we get

$$\begin{aligned} Max - Stretch^{\sigma} &= max_{j\in[1..log_2(c)]} Max - Stretch^{\sigma_j} \\ &\leq p_{max}(J) \leq Max - Stretch^{\sigma^*} \end{aligned} \tag{4}$$

From Equation 3, summing over $j \in [1..log_2(c)]$, we get

$$Total - Stretch^{\sigma} = \sum_{j\in[1..log_2(c)]} \sum_{J_i \in J^j} p_i = \sum_{J_i \in J} p_i \leq Total - Stretch^{\sigma^*} \tag{5}$$

□

Theorem 9 *Let $c > 1$ be a constant, J be any job sequence such that $p_{max}(J)/p_{min}(J) \leq c$, Λ be an upper bound on $N_{min}(J)$, and a set S consisting of a linear or circular list of $N \geq 2\Lambda c$ processors, then $Online-Allocator(J, N, S)$ is an $O(1)$-speed-1-competitive algorithm for the $Max-Stretch$ and $Total-Stretch$ problems.*

The proof of this theorem is similar to Theorem 8 so we skip the proof.

Theorem 10 *Given some constant $c \geq 1$, a job sequence J such that $p_{max}(J)/p_{min}(J) \leq c$ and a set S consisting of a linear or circular list of $N = 8log_2(c)N_{min}(J)$ processors, the algorithm $H(J, N, S)$ is an $O(1)$-speed-1 competitive algorithm for the Max-Stretch and Total-Stretch problems.*

Proof Let $c \geq 1$ be some constant, $J = (J_1, ..., J_n)$ be a sequence of n jobs such that $t_{max}(J)/t_{min}(J) \leq c$, and $N_H = 8log_2(c)N_{min}(J)$. Let OPT denote any optimal offline algorithm that minimizes $TotalStretch$ or $MaxStretch$ for J. For the job sequence J, let $S^{OPT} = (P_1, ..., P_{N^{OPT}})$ be the linear or circular list of N^{OPT} processors available to OPT, σ^* be the assignment determined by OPT, $S^H = (P_1, ..., P_{N^G})$ be the linear or circular list of N^H processors available to H and σ be the assignment determined by H.

Notice that algorithm H executes in phases and during each phase executes $General-Online-Allocator$. Since all jobs in J are allotted processors contiguously we know that H obtains an optimal solution for both the $MaxStrtch$ and $TotalStretch$ problems. So to establish this theorem, we only need to show that H uses at most $O(1)$ times more processors than the OPT in order to obtain an optimal solution.

Let h be the phase during which H completes assigning processors to jobs in J. If $h = 1$, then by Theorem 10 we are done. Otherwise, since H has entered phase h it must have exited phase $h - 1$. Now, we will get a bound on N_{OPT}. Notice that H assigns processors to jobs in $J^{(h-1)}$ during phase $h - 1$. Since H exited phase $h - 1$, we know from Theorem 10 $N_{OPT} \geq N_{min}(J^{(h-1)}) \geq N_{est}^{h-1} = 2^{h-1}p_{max}(J)$. Now, we will get an upper bound on N_H. We can observe that N_H equals the sum the sum of the processors assigned from phase 1 to phase h. Therefore, $N_H = \sum_{i \in [1..h+1]} c2^i p_{max}(J) \leq c2^{h+2}p_{max}(J)$. Now, we get $N_H/N_{OPT} = c2^{h+2}p_{max}(J)/2^{h-1}p_{max}(J) \leq 8c$. Hence the result. $\square$

7 Conclusions and Future Work

In this paper, we present offline and online algorithms that obtain optimal solutions using $O(1)$ more resources than the optimal offline algorithm when the processing times are bounded or when the number of required processors are bounded. For the offline problem, we have been able to exploit some of the properties of processor assignment mentioned in Section 5 to obtain stronger results. However, for the online case our recent results indicate that our online result is within a constant factor of the optimal. In view of these results, we are exploring alternate models where jobs can be refused.

Acknowledgments

I would like to thank the Algorithms Reading Group at State University of New York, Stony brook for bringing this open problem to my attention and in particular, I would like to thank Professors Michael Bender and Krishnan Pillaipakkamnatt for many useful discussions. I would also like to thank Hofstra University for supporting part of this research through their grant HRDG-2004.

References

[1] Esther Arkin, Michael Bender, David Bunde, Alok Lal, Vitus J. Leung, Joseph S.B. Mitchell, Cindy A. Phillip and Steve Seiden, Methods for Improving CPlant through Processor Allocation, *Personal Communication with Professor Bender*, 2002.

[2] Esther Arkin, Michael Bender, David Bunde, Alok Lal, Vitus J. Leung, Joseph S.B. Mitchell, Cindy A. Phillip and Steve Seiden, Processor Allocation on Cplant: Achieving General Processor Locality using One-Dimensional Allocation Strategies, *Sandia National Laboratory Report SAND2002-1488*, 2002.

[3] Borodin, A. and El-Yaniv, R. Online Computation and Competitive Analysis, *Cambridge University Press*, 1998.

[4] R. Brightwell, H.E. Fang, and L.Ward. Scalability and performance of CTH on the Computational Plant. *In Proc. 2nd International Conference on Cluster-Based Computing*, 2000.

[5] Chandra Chekuri and Sanjeev Khanna, On Multi-dimensional Packing Problems, *In Proc. 10th Annual ACM SIAM Symposium on Discrete Algorithms (SODA)*, pages 185-194, 1999

[6] F.R.K. Chung, M.R. Garey, and D.S. Johnson. On packing two-dimensional bins. *SIAM Journal on Algebraic and Discrete Methods*, 3:66-76, 1982.

[7] E.G. Coffman, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 2. PWS Publishing Company, 1997.

[8] D. Coppersmith and P. Raghavan. Multidimensional online bin packing: Algorithms and worst case analysis. *Operations Research Letters*, 8:17-20, 1989.

[9] J. Csirik and G. Woeginger. On-line packing and covering problems. In A.Fiat and G.Woeginger, editors, *On-Line Algorithms The State of the Art*, Lecture Notes in Computer Science, chapter 7, Springer-Verlag, 1998.

[10] S. Divakaran, Approximation Algorithms for a Resource Allocation Problem, *13th International Conference on Combinatorial Optimization*, CO2004, Lancaster, U.K., 2004.

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

[12] Kalyanasundaram, B. and Pruhs, K., Speed is as powerful as clairvoyance, *Proceedings of 36th Annual Symposium on Foundations of Computer Science* 214-225, 1995.

[13] Y. Kohayakawa, F.K. Miyazawa, P. Raghavan, and Y. Wakabayashi. Multidimensional cube packing. *In Electronic Notes of Discrete Mathematics* , volume 7. Elsevier Science, 2001. Presented at the Brazilian Symposium on Graphs and Combinatorics.

[14] M.G. Luby, J.S. Naor and A. Orda. Tight bounds for dynamic storage allocation. *SIAM Journal on Discrete Mathematics*, 9(1):155-166, 1996.

[15] J.M. Robson. An estimate of the store size necessary for dynamic storage allocation. *Journal of the ACM*, 18(3):416-423, 1971.

[16] Sandia National Laboratories, The Computational Plant Project, www.cs.sandia.gov//cplant.

[17] S. Seiden and R. van Stee. New Bounds for multi-dimensional packing. *In Proc. 13th Annual ACM SIAM Symposium on Discrete Algorithms*, (SODA), pages 486-495, 2002.

[18] J. Sgall. On-line scheduling. In A.Fiat and G. Woeginger, editors, *On-Line Algorithms The State of the Art*, Lecture Notes in Computer Science, chapter 9, Springer-Verlag, 1998.

[19] V. Subramani, R. Kettimuthu, S. Srinivasan, J. Johnson, and P. Sadayappan. Selective buddy allocation for scheduling parallel jobs on clusters, *In Proc. 4th IEEE International Conference on Cluster Computing*, 2002.