TR-IIS-09-006

Data-bandwidth-aware Job Scheduling Techniques in Distributed Systems

De-Yu Chen, Pangfeng Liu, Jan-Jan Wu

June 17, 2009 || Technical Report No. TR-IIS-09-006
http://www.iis.sinica.edu.tw/page/library/LIB/TechReport/tr2009/tr09.html

Data-bandwidth-aware Job Scheduling Techniques in Distributed Systems

De-Yu Chen
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan
r96083@csie.ntu.edu.tw

Pangfeng Liu
Department of Computer Science and Information Engineering
Graduate Institute of Networking and Multimedia
National Taiwan University
Taipei, Taiwan

Jan-Jan Wu
Institute of Information Science
Academia Sinica
Nankang, Taiwan

Abstract

This paper introduces techniques in scheduling jobs on a master/workers platform where the bandwidth is shared by all workers. The jobs are independent and each job requires a fixed amount of bandwidth to download input data before execution. The master can communicate with multiple workers simultaneously, provided that the bandwidth used by the master and the workers do not exceed their bandwidth limits.

We proposed two models for this limited-bandwidth problem. If the data transfer cannot be interrupted, then we prove that the scheduling problem is NP-complete. Nevertheless we propose heuristic algorithms and experimentally test their performance. If the data transfer can be interrupted, we propose an algorithm that produces optimal makespan. The algorithm is based on a binary search on the completion time, and an efficient feasibility verification process for a given completion time.

1 Introduction

Grid computing is becoming more and more popular in both academic and industry recently. In grid systems, distributed heterogeneous resources are connected through local and/or wide area networks. By utilizing the vast amount of computing and storage resources in grid systems, many large scale resource-demanding problems can be solved within a reasonable amount of time. For example, the Worldwide Large Hadron Collider Computing Grid [9] builds and maintains a data storage and analysis infrastructure for the entire high energy physics community. Other organizations like the Biomedical Informatics Research Network [5] provides a platform for biomedical scientist to share data and computing resources.

Many applications running on grids are for research purpose, and most of them take large amount of input data and perform complex computation to produce useful information. However, grid systems are often built across the wide area network and often consist of many sites distributed around the world. The cost to transfer input data thus plays an important rule to the overall efficiency of the application. Due to the heterogeneous nature of the wide area network, the computing sites in grid systems often differ from each other on their computing and communication capabilities. As a result, scheduling jobs in grid systems is always an important issue.

In this paper, we concentrate on scheduling jobs on a master/workers platform, where all jobs are initially placed on the master processor and need to be dispatched to workers for execution. We adopt the bounded multi-port model [8]. In this model, the master can communicate with multiple workers simultaneously, provided that the bandwidth used by the master and the workers do not exceed their bandwidth limits.

The rest of the paper is organized as follows. In Section 2, we begin by reviewing related works on scheduling in system in which computing nodes have communication bandwidth constraints. In Section 3, we describe our system model. We describe two variations of our problem in detail in Section 4 and Section 5. Finally, we conclude our works in Section 6.

2 Related Works

Scheduling computational tasks on a given set of processors is a key issue for parallel and distributed systems. General scheduling problems on multi-processor system has been showed to be NP-complete [11]. Even if there are only two processors with identical computing capacity and jobs do not share data, the problem remains NP-complete because it is a special case of 2-Partition problem [7].

Most applications deployed in grid systems require input data [6, 2]. The *Divisible Load Scheduling* model [3] assumes that the work for an application can be arbitrarily divided into any number of "chunks", where each chunk consists of some amount of input data and some computation to perform on this data. The problem is proved to be NP-complete in [12] by Yang et. al. (??? result from Divisible Load Scheduling here) In [4], Beaumont et. al. also targeted the bounded multi-port model. The main difference between our works and the work of Beaumont et. al. [4] is that they adopted the divisible load model, where both work and bandwidth can be divided arbitrarily, while in our model the tasks are not divisible and the bandwidth can only be divided into pieces of unit sizes. (??? is this correct?)

Bags-of-tasks [1] is another popular model for scheduling tasks on parallel and distributed systems. In Bags-of-tasks model, an application is a collection of *independent* and *identical* tasks. In [10], Legrand et. al. studied the scheduling of bags-of-tasks on master-worker platforms. They also assumed a multi-port model which is similar to our communication model. However, in their model the bandwidth of the master is shared fairly by all ongoing communications. On the other hand we focus on allocating bandwidth for tasks so that the the entire execution time is minimized in our model.

Master/workers model is freqently adopted for scheduling tasks on grid systems.

3 System Model

We consider a grid system as a collection of a master processor M_0 and n workers processors $M_1, \ldots, M_n$. All the processors are connected by a logical *star topology*, with the master in the center.

Each processor M_i has a communication bandwidth bound B_i, which limits the bandwidth it can use to send/receive data to/from other processors, and processors may have different communication bandwidth bounds. We assume that all B_i are integers so that we can divide the bandwidth into pieces of unit sizes. In addition we assume that the master can communicate with multiple workers simultaneously, as long as the total communication bandwidth used by the master does not exceed the master's bandwidth bound B_0.

3.1 Jobs

There are n jobs to process and worker M_i will process job J_i, for i from 1 to n. A task is for processor to perform computations on its input data. Initially all jobs are placed on the master and will be dispatched to workers for computation.

Job J_i has two integer attributes – input data size D_i and computation time T_i. The data size D_i is measured in terms of the amount of data that can be sent using one unit of bandwidth in one unit of time. At any given time step the communication bandwidth used by the master to send data to worker J_i can be an arbitrary integer not exceeding the bound B_i. For example, if D_i is 6, then job D_i can finish its communication by allocating two units of bandwidth in the current unit time step, and four units of bandwidth in the next unit time step, assuming its communication bandwidth bound M_i is at least four.

The computation time T_i is measured in terms of unit time steps for J_i to complete its computing. Note that a worker has to wait for all of its input data before it can start a job, and after the worker collects all of its input data, the computation will finish in T_i unit time steps, regardless the situation on other processors. Note that our model can also apply to systems in which processors have different computation abilities since the time T_i is defined as the time for M_i to execute J_i. We also assume that the master processor does not perform any computation because it is only responsible for sending input data to workers.

3.2 Problem Definition

A *bandwidth block* b_{ij} is the i-th unit bandwidth the master can provide at time t, and there are M_0 of them for every time step. We can think of these "blocks" as commodities that the master can provide M_0 of them per time step. Please refer for Figure 1 for an illustration. A schedule Ψ is a function that maps each bandwidth block to a job. In other words, a schedule determines how the master allocates its communication bandwidth among jobs at every time step.

Given a schedule Ψ we can determine *ready time* and *completion time* for all jobs. A job J_i is *ready* at time k when it has accumulated enough bandwidth for its data size D_i, i.e., till time k the mapping function Ψ has mapped D_i *bandwidth blocks* for J_i. Therefore the ready time of a job J_i under Ψ is the earliest time when job J_i is ready. We use $R_i(\Psi)$ to denote the ready time of J_i under Ψ. A worker begins processing its job immediately after the job is ready. Consequently the completion time of job J_i is the sum of its ready time $R_i(\Psi)$ and its computation time T_i. We use $C_i(\Psi)$ to denote the completion time of J_i under Ψ.

The *makespan* of a schedule is defined as the maximum completion time of all jobs, denoted by $C(\Psi)$. Our goal is to find a schedule that minimizes the makespan. This schedule will be refereed to as as the *optimal schedule*, denoted by Ψ^*. We also use C^* to denote this minimum possible makespan, or *optimal makespan*.

Figure 1 illustrates an example of schedule. Each colored blocks indicates the bandwidth blocks allocated to a job, and Each dotted lines at the end of a colored block represents the computation phases of that job. For example, at the first time step the master allocates two blocks to J_1, two blocks to J_2, and one block to J_3. At time 6 J_1 becomes ready since the master has already allocated $D_1 = 12$ blocks to it, so that J_1 can start computing. The computing ends at time 11 since the computing time T_1 is 5, and the ready time $R_1(\Psi)$ is 6, so the completion time $C_1(\Psi)$ is 11.

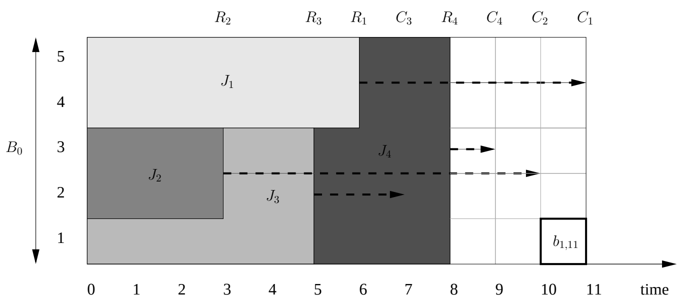

Figure 1. An example of schedule

We formally define the problem Limited-Bandwidth-Scheduling (LBS) as follows.

Definition 1 (LBS). *Given the bandwidth bounds of a master processor M_0 and n worker processors $M_1, \ldots, M_n$, a set of n jobs J_i, and the data size D_i and computing time T_i, find a schedule Ψ that minimizes the makespan.*

A job J_i is *interrupted* in a schedule Ψ if there exist time steps $t_1 < t_2 < t_3$ such that Ψ assign bandwidth blocks to J_i at time t_1 and t_3, but none at time t_2. In practice certain applications do not allow data transfer to be interrupted, therefore we formulate the uninterrupted variation of Limited-Bandwidth-Scheduling.

Definition 2 (LBS-Uninterruptible). *Given the bandwidth bounds of a master processor M_0 and n worker processors $M_1, \ldots, M_n$, a set of n jobs J_i, and the data size D_i and computing time T_i, find a schedule Ψ that minimizes the makespan and no job was interrupted.*

3.3 Notations

Table 1 summarizes the notations in our system model.

Notation	Description
M_0	the master processor
M_i	i-th worker processor
B_0	outgoing bandwidth bound of the master
B_i	incoming bandwidth bound of i-th worker
J_i	job for i-th worker
D_i	input data size of J_i
T_i	computation time of J_i
Ψ	a schedule
Ψ^*	an optimal schedule
b_{ij}	i-th unit of communication bandwidth of the master in j-th time slot
$R_i(\Psi)$	ready time of J_i under Ψ
$C_i(\Psi)$	completion time of J_i under Ψ
$C(\Psi)$	makespan of Ψ
C^*	the optimal makespan

Table 1. Notations used in the system model

4 Scheduling with Uninterruptible Communication

In this section, we study a variation of limited-bandwidth-scheduling in which the data transfer of jobs cannot be interrupted. We refer to this problem as LBS-Uninterruptible. We first prove that LBS-Uninterruptible is NP-complete, then we describe a special case of LBS-Uninterruptible in which an optimal solution can be easily found. Finally, we propose heuristic algorithms that can find efficient schedules for LBS-Uninterruptible.

4.1 NP-Completeness

Theorem 1. *The uninterruptible limited-bandwidth-scheduling problem is NP-complete.*

Proof. It is trivial that uninterruptible limited-bandwidth-scheduling problem is in NP since a non-deterministic Turing machine can easily determine the mapping function, and verify the answer in polynomial time.

We prove that uninterruptible limited-bandwidth-scheduling problem is NP-hard by reducing from 2-Partition [7]. Let $I = (a_i)_{1 \le i \le n}$ be an instance of 2-Partition, such that $\sum a_i = 2A$. A solution to this instance is a partition of the a_i's into two groups G_1 and G_2 such that $\sum_{i \in G_1} a_i = \sum_{i \in G_2} a_i = A$. We build an instance I' of LBS-Uninterruptible with two workers and n jobs, where $B_1 = B_2 = 1$, $B_0 = 2$, $D_i = a_i$, and $T_i = 0$.

If there is a solution for the 2-Partition instance I, we can easily construct a schedule for LBS-Uninterruptible I' with makespan of A. We reserve one row of bandwidth blocks for jobs in G_1, and the other row of bandwidth blocks for jobs in G_2. The job sequence in the first row is arbitrarily taken from G_1 as long as consecutive blocks are allocated to the same job. We repeat this process for the second row and we have a solution for LBS-Uninterruptible problem instance I'

We observe that since the bandwidth bound for every job is 1, and the data transfer cannot be interrupted, we can allocate a time interval of length D_i in which the schedule maps exactly one block to J_i for every time step in this interval. Also notice that master must allocates exactly *two* blocks to jobs in order to meet the time bound.

Now if we have a solution for LBS-Uninterruptible problem instance I', we can derive a solution for the 2-Partition instance I. We derive this by mapping the time interval windows of jobs to either "upper" or "lower" row of bandwidth blocks. The rule is that if two windows overlap, then they must be assigned differently as "upper" and "lower". We first arbitrarily pick one of the two jobs that starts at time 0 as "upper", and the other as "lower". Without lose of generality we assume that the "upper" job J has a longer computing time. Now we assign those windows that overlap with J as "lower". Eventually one such job J' will have a larger ready time than J, then we repeat this process on J', and assign every jobs that overlap with J' as "upper". Since every job must be a window and there is no "hole" in the schedule, we can repeat this process until all jobs are assigned. This will be a solution for the 2-Partition instance I since the sums of window lengths from both the upper and lower rows are both A. □

4.2 Unlimited Model

Despite the NP-completeness of general LBS-Uninterruptible problem, we can still find optimal solutions for LBS-Uninterruptible in special cases. For example, we will consider a special case in which the communication bandwidth bound of each worker is at least the communication bandwidth bound of the master, and derive the optimal solution. We will refer to this model as *unlimited model*.

We now define a *sequence schedule* in the unlimited model. Since the number of blocks that can be assigned to a job per time step is only limited to the bandwidth bound of the master, we can define a special class of scheduling that allocates *all* blocks of every time step to the first job until it has all the data, then allocates blocks to the second job, and so on. Therefore the scheduling can be described as a sequence of jobs, and the master will allocate blocks according to this sequence. Note that the sequence scheduling is possible in the unlimited model since there is no bandwidth bound placed on jobs.

We establish the following lemma, which states that we can always find an optimal schedule that is also a *sequence schedule*.

Lemma 1. *Given an problem instance in the unlimited LBS-Uninterruptible model, there exists an optimal schedule that is also a sequence schedule.*

Proof. Let Ψ^* be an optimal schedule. We will convert Ψ^* into a sequence schedule without increasing the makespan.

The transformation works as follow. We first sort jobs according to their ready time under the schedule Ψ^*, and denote the job that has i-th earliest ready time as J_1. If the master allocates bandwidth blocks to jobs other than J_1 before J_1 is ready (at time $R_i(\Psi^*)$), we switch these blocks with those allocated to J_1 at $R_1(\Psi^*)$. Note that this will not delay the ready time of J_1 since the blocks it has at $R_i(\Psi^*)$) will only be moved earlier in time. On the other hand, let J_i be the job that was allocated blocks before $R_1(\Psi^*)$, the ready time of J_1 under $R_1(\Psi^*)$. This switch will not delay the ready time of J_i either since the ready time of J_i is at least $R_1(\Psi^*)$, by the definition that J_1 has the earliest ready time. We repeat the switching until no block was allocated to jobs other than J_1 before J_1 is ready. After finishing moving all blocks of job J_1 forward we can move the blocks of J_2, and so on. Eventually we have a sequence schedule. Since we did not delay ready time for any job during the transformation, the makespan will not increase and the resulting sequence schedule is also optimal. □

With Lemma 1 in place we know that we can construct an optimal schedule for the unlimited model by focusing on only sequence schedules. That is we want to find a good sequence of jobs for the master to transfer data to. The following theorem establishes the optimal job sequence.

Theorem 2. *Given an instance of LBS-Uninterruptible in the unlimited model. the sequence schedule in which the master allocates blocks to jobs in the order of non-increasing computation time, has the minimum makespan.*

Proof. According to Lemma 1, there exist an optimal schedule Ψ^* where the master sends the jobs as a sequence π. Let J_{π_i} be the i-th job in this sequence π. We want to argue that if there are two consecutive jobs in π that are "out of order", i.e. the job with longer computing appears later in π, then we can switch them without increasing the makespan. If this is true then we can repeatedly switch those jobs that are out of order in π, and derive a sequence schedule Ψ' that is the same as the schedule that follows the computing time order, and without increasing the makespan. Since Ψ^* is already an optimal schedule, Ψ' is also an optimal schedule.

Let J_{π_i} and $J_{\pi_{i+1}}$ be two successive jobs in π, and J_{π_i} has a shorter computation time than $J_{\pi_{i+1}}$, i.e. $T_{\pi_i} < T_{\pi_{i+1}}$. Exchanging them in the allocation will not increase the makespan, as suggested by the following inequality. Note that Ψ' denotes the new schedule after exchanging J_{π_i} and $J_{\pi_{i+1}}$.

$$
\begin{aligned}
& \max(C_{\pi_i}(\Psi^*), C_{\pi_{i+1}}(\Psi^*)) && (1)\\
= \;& \max(R_{\pi_i}(\Psi^*) + T_{\pi_i}, R_{\pi_{i+1}}(\Psi^*) + T_{\pi_{i+1}}) && (2)\\
= \;& R_{\pi_{i+1}}(\Psi^*) + T_{\pi_{i+1}} && (3)\\
\geq \;& \max(R_{\pi_{i+1}}(\Psi^*) + T_{\pi_i}, R_{\pi_{i+1}}(\Psi') + T_{\pi_{i+1}}) && (4)\\
= \;& \max(R_{\pi_i}(\Psi') + T_{\pi_i}, R_{\pi_{i+1}}(\Psi') + T_{\pi_{i+1}}) && (5)\\
= \;& \max(C_{\pi_i}(\Psi'), C_{\pi_{i+1}}(\Psi')) && (6)
\end{aligned}
$$

Equation 3 holds because $J_{\pi_{i+1}}$ has a later ready time and a longer execution time than J_{π_i} in Ψ^*. Inequality 4 holds because $T_{\pi_{i+1}}$ is longer than T_{π_i}, and the ready time of $J_{\pi_{i+1}}$ was moved earlier in Ψ'. Finally it is easy to see that the ready time of $J_{\pi_{i+1}}$ in Ψ^* is the same as the ready time of J_{π_i} in Ψ', i.e., $R_{\pi_{i+1}}(\Psi^*) = R_{\pi_i}(\Psi')$, therefore we have Equation 5. $\square$

4.3 Heuristic Algorithms

In this section, we propose heuristic algorithms that find good schedules for LBS-Uninterruptible problem. The heuristic is inspired by the optimal algorithm for unlimited model from Theorem 2. We first sort the jobs in according to a given priority metric, then construct a schedule by allocating as many bandwidth blocks as possible to the job that has the highest priority, until all of it data is avail and it is ready for execution. By adopting this greedy scheme, various schedules can be constructed based on different priority metric. For example the optimal algorithm for unlimited model uses non-decreasing computing time order.

4.3.1 Non-Increasing T_i

The first priority metric we tested is the same non-increasing computing time order we used in Section 4.2 to construct an optimal schedule for the unlimited model. A job with a long computation time is likely to has a large completion time and thus is more likely to increase the makespan, therefore by it ready as early as possible might be helpful in reducing makespan.

4.3.2 Non-Increasing D_i/B_i Ratio

The general case of LBS-Uninterruptible differs from the unlimited model in Section 4.2 in the bandwidth bounds of the workers. If a job has a large D_i/B_i ratio, i.e. it requires a large amount of data but it has only limited bandwidth to download them, then it is very likely its ready time will be delayed. As a result we should schedule such jobs as early as possible.

4.3.3 Non-Increasing $D_i/B_i + T_i$

The total processing time of a job is the summation of its data downloading time and computing time, and the data downloading time of J_i is about D_i/B_i if the bandwidth is not shared by other jobs. $D_i/B_i + T_i$ thus may be a good approximation for the total processing time of J_i, and a job with larger total processing time should be scheduled earlier in order to reduce the makespan. ??? How?

4.4 Performance Evaluation

In this section we compare the performances of the heuristics. We run the heuristics on problem sizes for which we can use exhaustive search to compute the optimal makespans. Then we compute the performance of the heuristic algorithm against the optimal solution.

In the first set of experiments we set the bandwidth limit $B_0 = 10$ for the master, and B_i's randomly from 1 to 10 for the workers. For each job J_i, we set the data requirement D_i randomly from 1 to 15 and the execution time T_i randomly from 1 to 10. Figure 2 shows that by setting the priority metric to be non-increasing $D_i/B_i + T_i$, we can construct schedules with almost optimal makespan, even when the number of jobs is increasing.

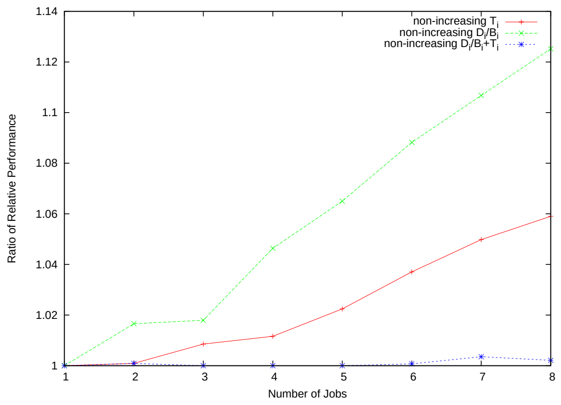

Figure 2. Relative performances of the heuristics.

In the second set of experiments we try tighter bandwidth bounds of the processors. In particular we set the bandwidth limit $B_0 = 5$ for the master and B_i randomly from 1 to 5 for workers. Figure 3 shows that while the non-increasing $D_i/B_i + T_i$ metric is still performs best, all heuristics perform worse than they did under larger bandwidth bounds as in Figure 2.

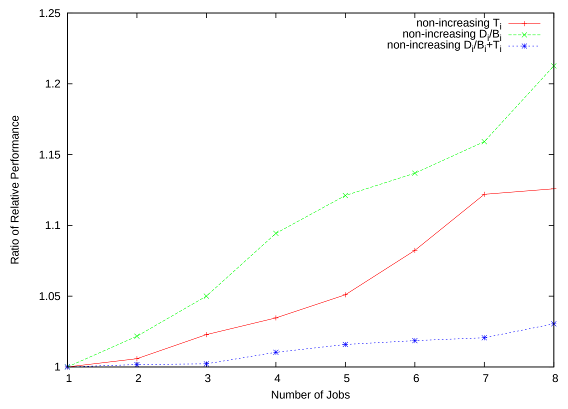

Figure 3. Relative performances of the heuristics.

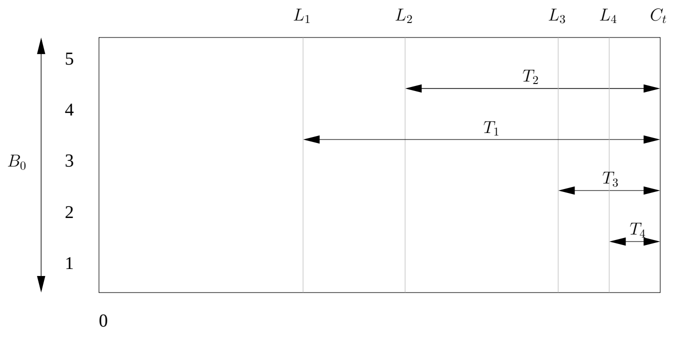

Figure 4. An illustration of C_t and L_i's.

5 Scheduling with Interruptible Communication

in this section we study the general limited-bandwidth-scheduling (LBS) problem. Unlike the uninterruptable version we discussed in Section 4, now the the data transfers of jobs *can* be interrupted. While uninterrupted version of limited-bandwidth-scheduling is NP-Complete, we can find optimal schedule for limited-bandwidth-scheduling if now interruption in data transfer is allowed. We will first describe our algorithm in details, then prove that the schedule constructed by the algorithm is indeed optimal.

5.1 The Algorithm

Our limited-bandwidth-scheduling algorithm works in phases and each phase has two steps. In the first step we "guess" a completion time. In the second step, we verify that the guessed completion time is feasible or not. By a binary search on the completion time, where each phase in our algorithm is a step in the binary search, we will be able to find the *minimum* completion time while there still does exist a feasible schedule.

5.1.1 Completion Time Guessing

We now describe the first step for each phase of the algorithm – completion time guessing. Before actually allocating bandwidth blocks to jobs, we guess a completion time C_t for the entire execution. Once we have a C_t for our schedule, the latest ready time L_i for each job J_i can be derived as $L_i = C_t - T_i$. L_i is the deadline for job J_i to start execution, otherwise the completion time C_t cannot be accomplished. Figure 4 gives an illustration of C_t and L_i's.

5.1.2 Feasibility Verification

Given the deadlines for the jobs to collect all the data and start execution, we verify if we can allocate bandwidth to jobs so that every job can meet its deadline. The main idea of our bandwidth allocation process is to keep the allocated bandwidth blocks as "flat" as possible at all time. That is, we will use roughly the same amount of bandwidth at all time.

We first determine the order of bandwidth allocation. Without lose of generality we assume that the jobs $J_1, \ldots, J_n$ are indexed according to a non-decreasing order of their deadlines, i.e. $L_1 \leq \cdots \leq L_n$. The algorithm will first allocate bandwidth for the first job J_1, then for the next job J_2, and so on, until all jobs are allocated enough bandwidth. If at any time the algorithm finds that it cannot allocate enough bandwidth before the deadline of the current job, it report failure and we know the current guessed completion time is not feasible.

We allocate bandwidth to the jobs as follows. For the purpose of correctness proof we assume that the algorithm allocates *one* bandwidth block at a time for a job J_i until it gets enough bandwidth to transfer its input data. The algorithm allocates a bandwidth block b_{jk} to J_i if the allocation of b_{jk} satisfies the following three conditions.

- b_{jk} is not yet allocated to any job.
- The block b_{jk} is earlier than L_i, i.e. $k \leq L_i$.
- Allocating b_{jk} to J_i does not violate the bandwidth constraint of J_i, i.e. the number of blocks at time k that have been allocated to job J_i is no more than the bandwidth limit B_i of job J_i.

If no such bandwidth block can be found, then the algorithm returns failure and we conclude that no feasible schedule with completion time C_t can be found. If there are more than one bandwidth blocks which satisfy all of the conditions, we choose the one with *smallest* j among those bandwidth blocks. If there are more than one bandwidth blocks with the same smallest j, choose the one with smallest k. In other words, we first allocate blocks in the row with the minimum row index, and within each row we select blocks in increasing time order until the deadline or the bandwidth constraint is violated. We will refer to this allocation as "min-row-first" method. Please refer to Figure 5 for an example.

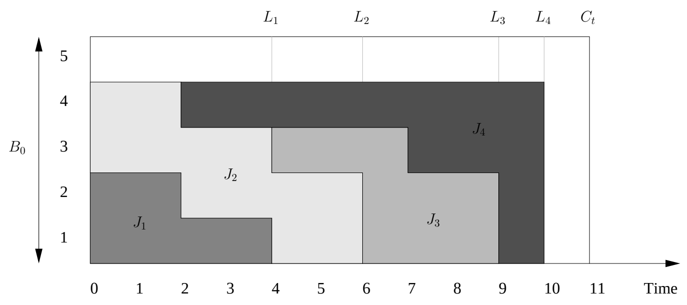

Figure 5. An example of min-row-first scheduling.

If we can allocate enough blocks for every job with the min-row-first method, we have a feasible schedule. The feasibility of the resulting schedule can be easily checked since we only allocate bandwidth blocks within the bandwidth constraint and every job meets its data collection deadline.

If we cannot allocate enough blocks for any job using the min-row-first method, we want to conclude that there will be *no* feasible schedule under the current completion time assumption. That is, we want to prove that the min-row-first can find an optimal schedule for limited-bandwidth-scheduling problem, if one does exist.

Theorem 3. *There exists a feasible schedule Ψ with makespan $C(\Psi)$ if and only if the min-row-first algorithm return success with completion time set to $C(\Psi)$.*

Proof. We have already showed in Section 5.1.2 that the schedule returned by our algorithm is always feasible, so we only need to prove the "only if" part of the theorem by showing that given an arbitrary schedule Ψ with makespan $C(\Psi)$, we can transform it into the schedule Ψ' return by the min-row-first algorithm, when the completion time C_t is set to be $C(\Psi)$.

An important observation in the block bandwidth model is that there is *no* difference between blocks from the *same* time step. As long as blocks are from the same time step, it does not matter which block is allocated to which job. For ease of explanation we assume that those allocated to jobs with smallest job index will appear in rows with smallest index. We will refer to this convention as the *early-job-lower-row* convention. Figure 6 illustrates the same schedule in Figure 1 following this assumption. From now on we will assume that all the scheduling will follow this convention.

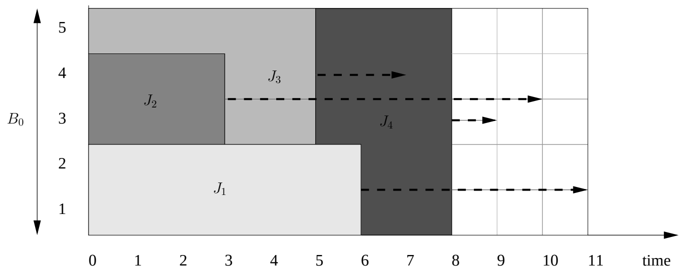

Figure 6. A schedule that puts job with smaller index into rows with smaller index (early-job-lower-row convention)

We consider two schedules – Ψ and Ψ', where Ψ is any feasible schedule for completion time $C(\Psi)$, and Ψ' is the schedule produced by the min-row-first algorithm under completion time $C(\Psi)$. We will show that we can always convert Ψ into Ψ' without increasing the completion time.

We will convert the allocation one job at a time in order as $J_1, \dots, J_n$, where we assume the jobs are indexed according to a non-decreasing order of their deadlines, i.e. $L_1 \leq \cdots \leq L_n$. Without lose of generality we assume that we are converting blocks allocated to J_i, and both schedules follows the convention that smaller indexed jobs go to smaller indexed rows.

We now focus on the "first" block that was allocated to J_i by Ψ' but was allocated to another job by Ψ. We assume that this block is b_{rt}. Now consider the block that was allocated to J_i by Ψ that has the largest row index, or the largest time step if there are more than one block allocated to J_i at that row. Let this block be $b_{r't'}$. We conclude that $r' \geq r$. That is, block $b_{r't'}$ is at a row with a larger or equal index than b_{rt} since Ψ' is the result of the min-row-first algorithm.

We consider those blocks at a time step that are assigned to jobs other than $J_1, \dots, J_i$ by Ψ, and refer to them as *free blocks* at that time step. Let S_t be the set of free blocks at time t and $S_{t'}$ be the set of free blocks at time t'. Since r' is no less than r, the number of free blocks at time step t is at least one larger than the number of free blocks in time step t', i.e. $|S_t| \geq |S_{t'}| + 1$. The reason is that because we follow the early-job-lower-row convention, and the fact that Ψ allocates at least one more blocks to jobs $J_1, \dots, J_i$ at time step t' than at time step t.

We now want to switch the allocations for blocks b_{rt} and $b_{r't'}$ in Ψ, so the allocations becomes the same for this block. We claim that there must exist a job J_k that was allocated at least a block in S_t, and the same job J_k was allocated at most $B_k - 1$ blocks in $S_{t'}$. If this is not the case, then every job J_k that has been allocated blocks from S_t will have B_k blocks in $S_{t'}$. which is impossible since the size of S_t is at least $|S_{t'}| + 1$ from previous discussion. As a result we switch the allocation so that Ψ allocates b_{rt} to job J_i and $br't'$ to J_k. The new allocation will not violate the bandwidth constraint for J_i since b_{rt} was allocated to Ψ', and there are at most $B_k - 1$ blocks allocated to J_k before the switch. Please refer to Figure 7 for an example.

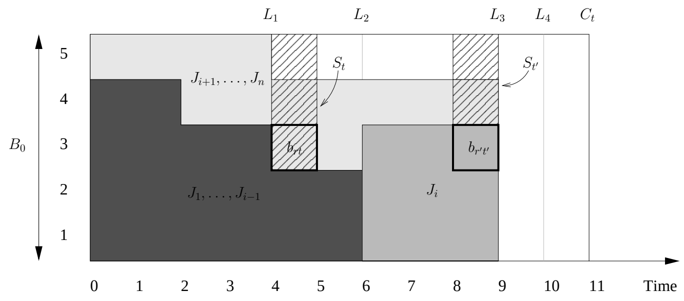

Figure 7. An illustration of b_{rt}, $b_{r't'}$, S_t and $S_{t'}$.

After establishing that we can switch b_{rt} and $b_{r't}$, we can repeat this process until Ψ and Ψ' becomes the same. The reason is that if Ψ and Ψ' differ, then there must be a $b_{r't'}$ that is located at the same or lower row, since the number of the blocks in Ψ and Ψ' are the same, and Ψ' uses the min-row-first algorithm to allocate blocks. The theorem follows. $\square$

With Theorem 3 in place we are certain that if there exists a feasible schedule with a given completion time, the min-row-first algorithm is able to find it. Therefore by a binary search on the completion time we will be able to find the minimum completion time while there still does exist a feasible schedule. However, we need to specify the upper bound for starting the binary search. It is easy to see that $C_t = \sum_{i=1}^{n} \lceil D_i / \min(B_0, B_i) \rceil$, is a trivial upper bound of the minimum completion time such that we can start the binary search.

6 Conclusions

This paper introduces techniques in scheduling jobs on a master/workers platform where the bandwidth is shared by all workers. The jobs are independent and each job requires a fixed amount of bandwidth to download input data before execution. The master can communicate with multiple workers simultaneously, provided that the bandwidth used by the master and the workers do not exceed their bandwidth limits.

We proposed two models for this limited-bandwidth problem. If the data transfer cannot be interrupted, then we prove that the scheduling problem is NP-complete. Nevertheless we propose heuristic algorithms and experimentally test their performance. If the data transfer can be interrupted, we propose an algorithm that produces optimal makespan. The algorithm is based on a binary search on the completion time, and an efficient feasibility verification process for a given completion time.

The authors would like to further investigate possible efficient algorithms in both models. Despite the fact that it is NP-complete to find the optimal solution when we do not allow data transfer interruption, it is still possible to find dynamic programming or approximation algorithm for the scheduling problem. Another possible future work is to improve the optimal algorithm for the model that allows interruption, since the current algorithm requires a binary search on the completion time.

References

[1] M. Adler, Y. Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *15th ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2003.

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[3] V. Bahradwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[4] O. Beaumont, N. Bonichon, and L. Eyraud-Dubois. Scheduling divisible workloads on heterogeneous platforms under bounded multi-port model. In *22nd IEEE International Parallel and Distributed Processing Symposium*, 2008.

[5] BIRN: The Biomedical Informatics Research Network. http://www.nbirn.net/.

[6] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The apples parameter sweep template: user-level middleware for the grid. In *Proceedings of the ACM/IEEE conference on Supercomputing*, pages 75–76, 2000.

[7] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman and Co. New York, NY, USA., 1979.

[8] B. Hong and V. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In *18th IEEE International Parallel and Distributed Processing Symposium*, 2004.

[9] LCG: LHC Computing Grid. http://lcg.web.cern.ch/LCG/.

[10] A. Legrand and C. Touati. Non-cooperative scheduling of multiple bag-of-task applications. In *26th IEEE International Conference on Computer Communications*, pages 427–435, 2007.

[11] J. D. Ullman. Np-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[12] Y. Yang, H. Casanova, M. Drozdowski, M. Lawenda, and A. Legrand. On the complexity of multi-round divisible load scheduling. Technical report, 2007.