アルゴリズム 46－9
（1995. 7. 20）

An Algorithm for Scheduling Jobs in Hypercube Systems

Oh-Heum Kwon and Kyung-Yong Chwa
Dept. Computer Science, KAIST
Kusong-dong 373-1, Yusong-gu, Taejon 305-701, Republic of Korea
{ohkwn,kychwa}@gayakum.kaist.ac.kr

In this paper, we consider the problem of non-preemptively scheduling independent jobs so as to minimize overall finish time on an m-dimensional hypercube system. This problem is NP-hard. We propose a polynomial time heuristic algorithm, and prove that the absolute performance ratio of the algorithm does not exceed 1.875. This is the first algorithm achieving an absolute performance ratio less than 2 by a constant. Moreover, we discuss other related problems including on-line scheduling problems.

An Algorithm for Scheduling Jobs in Hypercube Systems

Oh-Heum Kwon and Kyung-Yong Chwa
Dept. Computer Science, KAIST
Kusong-dong 373-1, Yusong-gu, Taejon 305-701, Republic of Korea
{ohkwn,kychwa}@gayakum.kaist.ac.kr

本文では，m 次元ハイパーキューブ上でのスケジューリング問題を解く多項式時間ヒューリスティックアルゴリズムを与え，アルゴリズムの性能比が 1.875 以下であることを証明する．なお，この問題は NP-困難であり，今までに性能比が 2 より小さい定数であるアルゴリズムは得られていない．また，オンラインスケジューリング問題等の関連する問題についても考察する．

1 Introduction

An m-dimensional hypercube (or m-cube) is an undirected graph $G = (V, E)$ with $V = \{0, \ldots, 2^m - 1\}$ and $(v, w) \in E$ iff the binary representations of v and w differ in exactly one bit. A subgraph G' of G is said to be a d-dimensional subcube (or d-subcube) iff G' is isomorphic to a d-cube. A hypercube system is a network of computers in which processors are located at the nodes of the hypercube and communication channels between processors are the edges of the hypercube. In such an environment, each job may just require a dedicated subcube and many jobs can be running simultaneously. Then the problem of how to assign a set of jobs to make efficient use of such systems becomes important [1, 7].

In this paper, we consider the problem of scheduling independent jobs so as to minimize the overall finish time (makespan) on an m-dimensional hypercube system. Each job $J_i, 1 \leq i \leq n$, is associated with a dimension d_i and processing time t_i, meaning that job J_i requires a d_i-subcube for t_i units of time. No preemption is allowed.

This problem is a generalization of the classical multiprocessor scheduling problem. So, it is NP-hard [4]. Accordingly, polynomial time algorithms whose solutions in the worst case are within a fixed multiplicative constant of the optimal solution have been investigated. For a set of jobs J, let $A(J)$ be the finish time of the schedule generated by an algorithm A, and let $OPT(J)$ be the optimal finish time. The *absolute performance ratio* [4] of algorithm A is defined to be the maximum ratio $\frac{A(J)}{OPT(J)}$ over all possible instances of input jobs J.

In [1], they proposed an algorithm called LDLPT(Largest Dimension Largest Processing Time) of which the absolute performance ratio is proved to be $2 - \frac{1}{2^{m-1}}$. Zhu and Ahuja [7] proved that a simple algorithm called LDF(Largest Dimension First) achieves an absolute performance ratio $2 - \frac{1}{2^m}$. These two algorithms are commonly based on the strategy called *list scheduling* [2] where the jobs are ordered by non-increasing dimensions.

In this paper, we propose another polynomial time approximation algorithm, and prove that the absolute performance ratio of the algorithm does not exceed $2 - \frac{1}{8}$. It is the first algorithm achieving an absolute performance ratio less than 2 by a constant.

2 Preliminaries

Let $G = (V, E)$ be an m-cube. For $a, b \in V$ with $a \leq b$, let $[a, b]$ denote the set of processors $\{p \in V | a \leq p \leq b\}$. We also call $[a, b]$ a *p-interval*, with a and b being the start point and end point of the p-interval, respectively. Also let $|[a, b]|$ be the number of processors in $[a, b]$.

Let $[u, v]$ be a p-interval, with $u = u_m \cdots u_1$ and $v = v_m \cdots v_1$ in binary. It is clear that $[u, v]$ forms a k-subcube iff i) $u_m \cdots u_{k+1} = v_m \cdots v_{k+1}$, ii) $u_k \cdots u_1 = 00 \cdots 0$, and iii) $v_k \cdots v_1 = 11 \cdots 1$. In that case, $[u, v]$ is said to be a *basic subcube*.

An arbitrary p-interval can be partitioned basic subcubes. For any p-interval I, a partition P of I into basic subcubes is said to be *minimal* if no two or more subcubes in P can be merged into a larger basic subcube. The minimal partition of any p-interval is uniquely determined. Let $MP(I)$ denote the minimal partition of p-interval I. For example, $MP([3, 17]) = \{[3, 3], [4, 7], [8, 15], [16, 17]\}$.

Let $J = \{J_i = (d_i, t_i) \mid 1 \leq i \leq n\}$ be a set of jobs to be scheduled. We will assume that $d_i \leq m - 1, 1 \leq i \leq n$. Note that this assumption makes no loss of the generality. The *size* of job $J_i = (d_i, t_i)$ refers the size of the required subcube, i.e., 2^{d_i}.

Let $t_{max}(J) = \max_{1 \leq i \leq n} t_i$, and let $h_{avg}(J) = \frac{1}{2^m} \sum_{1 \leq i \leq n} 2^{d_i} t_i$. $h_{avg}(J)$ is the average work which should be done by a processor. It is obvious that $OPT(J) \geq t_{max}(J)$ and that $OPT(J) \geq h_{avg}(J)$. In the followings, we will use h_{avg} and t_{max} instead of $h_{avg}(J)$ and $t_{max}(J)$, respectively.

Let $J_x \subseteq J$, where $0 \leq x \leq 1$, be the set of jobs with the processing time no less than $x \cdot t_{max}$, and let D_x denote the sum of the size of all jobs in J_x.

Since our scheduling algorithm depends heavily on LDF algorithm [7], we first give a summary on LDF algorithm. LDF algorithm is as follows: First, sort the jobs by the order of non-increasing size; let F be a set of basic subcubes; initially, $F = \{[0, 2^m - 1]\}$. Second, consider the jobs from the ordered list, one after another; when $J_i = (d_i, t_i)$ is considered, each subcube $[a, b] \in F$ is divided into $\frac{|[a,b]|}{2^{d_i}}$ basic d_i-subcubes, and then, J_i is assigned to the earliest available d_i-subcube in F.

Let $J_l = (d_l, t_l)$ be the job that finishes at time $LDF(J)$, and let s_l be the starting time of J_l. No processor is idle before s_l. This property was observed by Chen and Lai [1]. Hence, we have $LDF(J) = s_l + t_l \leq h_{avg} + t_{max} \leq 2OPT(J)$ — — — (1).

3 Scheduling Algorithm

In this section, we will give an algorithm that achieves an absolute performance ratio 1.875. This algorithm is a sort of hybrid one in the sense that it schedules the relatively long jobs firstly, in fact, the jobs in $J_{\frac{3}{4}}$, and then, schedules the remaining jobs by the similar method with LDF algorithm. That is, two principles – largest processing time first and largest dimension first – are used by a hybrid fashion.

In our algorithm, two different cases are considered separately. The first case is that $D_{\frac{3}{4}} \leq 2^{m-1}$. We give an algorithm called A for this case in Section 3.1. In Section 3.2, we give another algorithm called B for the second case of $D_{\frac{3}{4}} > 2^{m-1}$.

3.1 The case of $D_{\frac{3}{4}} \leq 2^{m-1}$

Algorithm A

Step 1 Let $J_{\frac{3}{4}} = \{J_1, J_2, \ldots, J_k\}$, where the jobs are indexed by the order of non-increasing sizes. The i-th job $J_i = (d_i, t_i), 1 \leq i \leq k$, is assigned to the p-interval $[\sum_{1 \leq j \leq i-1} 2^{d_j}, \sum_{1 \leq j \leq i} 2^{d_j} - 1]$ at time 0. Since the jobs are ordered by non-increasing size, the assigned p-interval always forms a basic subcube.

Step 2 Let $P = MP([0, D_{\frac{3}{4}} - 1]) \cup MP([D_{\frac{3}{4}}, 2^m - 1])$. Assume that each processor $p \in [0, D_{\frac{3}{4}} - 1]$ is available at time t_{max}. Let $J' = \{J'_1, J'_2, \ldots, J'_{n-k}\}$ be the set of remaining jobs that are also ordered by non-increasing size.

Step 3 Schedule each job in J', one after another. When $J'_j = (d'_j, t'_j)$ is considered, each subcube $[a, b] \in P$ with $|[a, b]| > 2^{d'_j}$ is divided into $\frac{|[a,b]|}{2^{d'_j}}$ basic d'_j-subcubes. J'_j is assigned to the earliest available d'_j-subcube in P.

Let $J_l = (d_l, t_l)$ be the job finished at time $A(J)$, and let s_l be the start time of J_l. If more than one jobs finish at time $A(J)$ simultaneously, choose the one considered first among them. Note that the jobs considered later than J_l do not increase $A(J)$ and also do not decrease $OPT(J)$. So, without loss of generality, we assume that J_l is the lastly considered job, which means that no job in $J - J_{\frac{3}{4}}$ has a dimension less than d_l.

First, consider a special case of $s_l \leq t_{max}$. Since $t_l \leq \frac{3}{4} t_{max}$, we have $A(J) = s_l + t_l \leq t_{max} + \frac{3}{4} t_{max} \leq \frac{7}{4} OPT(J)$, which is just what we want. Hence, in the followings, we will assume that $s_l > t_{max}$.

Let Q_A denote d_l-subcube $[j2^{d_l}, (j+1)2^{d_l} - 1]$ with $j2^{d_l} \leq D_{\frac{3}{4}} - 1 < (j+1)2^{d_l} - 1$. If $D_{\frac{3}{4}}$ is a multiple of 2^{d_l}, no such a subcube exists. In that case, let $Q_A = \emptyset$. We obtained Lemma 1.

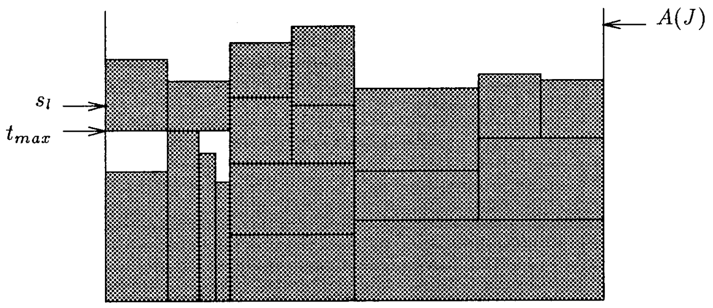

Figure 1: Example schedule where $Q_A = \emptyset$

Lemma 1 *In the schedule generated by algorithm A,*
(1) no processor $p \notin [0, D_{\frac{3}{4}} - 1] \cup Q_A$ is idle before s_l;
(2) no processor $p \in [0, D_{\frac{3}{4}} - 1] - Q_A$ is idle at any time t with $t_{max} < t < s_l$.

Proof. Suppose for contradiction that there is a processor $p \notin [0, D_{\frac{3}{4}} - 1] \cup Q_A$ which is idle at time t with $t < s_l$. Let I denote the subcube in $MP([D_{\frac{3}{4}}, 2^m - 1])$ which contains processor p. Since $p \notin Q_A$, the size of I is no less than 2^{d_l}. While excuting Step 3, I may be fragmented into the smaller basic subcubes. However, since each job considered prior to J_l in Step 3 has a dimension no less than d_l, it can not be fragmented into subcubes with size less than 2^{d_l}. As a result, job J_l should be assigned to the d_l-subcube including p at time t, which is a contradiction. In the case where $p \in [0 \cdots D_{\frac{3}{4}} - 1] - Q_A$, the similar argument can be applied. □

From Lemma 1, we can figure out the schedule generated by algorithm A as shown in Figure 1 and 2. Figure 1 shows an example schedule where $Q_A = \emptyset$, while Figure 2 shows another schedule where $Q_A \neq \emptyset$. Informally speaking, there are three types of wasted space in the schedule. We will call each of these wasted spaces *the internal waste, the external waste and the vertical waste*, respectively, as indicated in Figure 2. Note that Lemma 1 gives an upper bound on the area of vertical waste. Let's first consider the case where $Q_A = \emptyset$.

Lemma 2 *If $D_{\frac{3}{4}}$ is a multiple of 2^{d_l}, that is, $Q_A = \emptyset$, then $A(J) \le (2 - \frac{1}{8})OPT(J)$.*

Proof. As illustrated in Figure 1, there are only two types of wasted space: internal waste and external waste. We can easily obtain a lower bound on the total work of the jobs, that is, $2^m h_{avg}$, as follows: $2^m h_{avg} \ge A(J)2^m - \frac{3}{4}t_{max}2^m - \frac{1}{4}t_{max}2^{m-1}$, where the second and third terms of the right-hand side represent the external and internal waste, respectively. Lemma 2 follows immediately from this inequality. □

Now, consider the case of $Q_A \neq \emptyset$. In this case, we need to have a more tight lower bound on $OPT(J)$. To do this, we will define another set of jobs $J^\star$ by transforming the original set of jobs J. The transformation is as follows: First, replace the set of jobs $J_{\frac{3}{4}}$ in J by a set of $D_{\frac{3}{4}}$ equal jobs with the dimension zero and processing time $\frac{3}{4}t_{max}$. Second, each job $J_i = (d_i, t_i)$ in $J - J_{\frac{3}{4}}$ is partitioned into $2^{d_i - d_l}$ jobs with the dimension d_l and processing time t_i. Finally, insert a new set of $\lceil \frac{D_{\frac{3}{4}}}{2^{d_l}} \rceil 2^{d_l} - D_{\frac{3}{4}}$ jobs with the dimension zero and processing time $\frac{3}{4}t_{max}$. As a summary, the resulting set $J^\star$ includes two types of jobs: i) $\lceil \frac{D_{\frac{3}{4}}}{2^{d_l}} \rceil 2^{d_l}$ jobs with the dimension zero and processing time $\frac{3}{4}t_{max}$; ii) a set of jobs with the equal dimension d_l. Figure 3 shows the result of transformation of the job set in Figure 2.

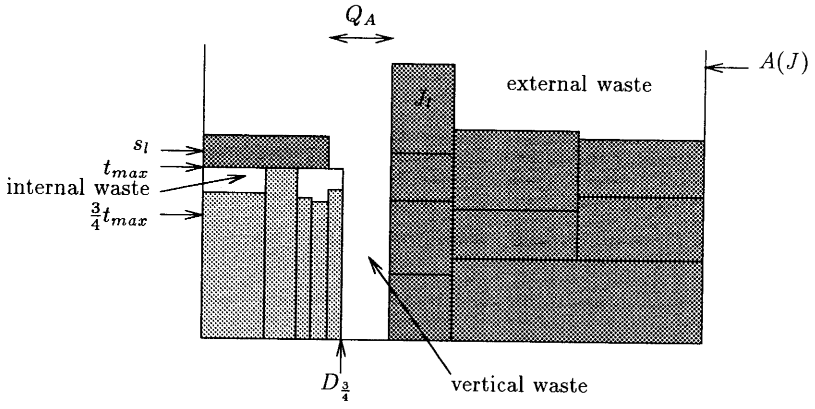

Figure 2: Example schedule where $Q_A \neq \emptyset$

Lemma 3 asserts that $OPT(J) \geq OPT(J^\star)$. It is obvious that the first and second steps of the transformation do not increase the finish time of the optimal schedule. Hence, it is sufficient to prove that the third step also does not increase the finish time of the optimal schedule. We will present the proof of Lemma 3 in Section 3.3.

Lemma 3 $OPT(J) \geq OPT(J^\star)$.

Now, we are ready to derive a lower bound on $h_{avg}(J^\star)$ in terms of $A(J)$: $2^m h_{avg}(J^\star) \geq 2^m A(J) - 2^{d_l}A(J) - (2^m - 2 \cdot 2^{d_l})\frac{3}{4}t_{max} - (2^{m-1} - 2^{d_l})\frac{t_{max}}{4} + \frac{3}{4}t_{max}2^{d_l}$ --- **(2)**. The second, third and fourth term of the right-hand side represent the vertical, external and internal waste, respectively. The fifth term corresponds to the jobs inserted in the third step of the transformation. From inequality (2), we can obtain Lemma 4.

Lemma 4 *For any set of jobs J with $D_{\frac{3}{4}} \leq 2^{m-1}$, if $A(J) > (2 - \frac{1}{8})OPT(J)$, then $LDF(J) \leq (1 + \frac{1}{6})OPT(J)$.*

Proof. In the case of $d_l < m-1$, it is straightforward to show that $A(J) \leq (2 - \frac{1}{8})OPT(J)$ from inequality (2). When $d_l = m-1$, inequality (2) is reduced to: $A(J) \leq 2OPT(J) - \frac{3}{4}t_{max}$. If $\frac{3}{4}t_{max} \geq \frac{1}{8}OPT(J)$, then we have $A(J) \leq (2 - \frac{1}{8})OPT(J)$; otherwise, from inequality (1), we have $LDF(J) \leq h_{avg} + t_{max} \leq (1 + \frac{1}{6})OPT(J)$. □

3.2 The case of $D_{\frac{3}{4}} > 2^{m-1}$

Algorithm B

Step 1 A subset J' of J is constructed as follows: Initially, $J' = \emptyset$; sort the jobs in J by non-increasing order of the processing times, and examine the jobs, one after another, in this order; for a job $J_i = (d_i, t_i) \in J$, if $t_i \geq \frac{1}{2}t_{max}$ and the sum of the size of jobs in $J' \cup \{J_i\}$ is no more than 2^m, then add J_i into J'.
Step 2 Sort the jobs in J' by non-increasing size, and schedule them into p-interval $[0, D-1]$ by the same method with Step 1 of algorithm A, where D is the sum of the size of all jobs in J'.
Step 3 Let $P = MP([D, 2^m - 1])$. The jobs in $J - J'$ are ordered by non-increasing size and considered one after another. When $J_j = (d_j, t_j)$ is considered, each subcube in P with the size larger than 2^{d_j} is partitioned into basic d_j-subcubes; the earliest available d_j-subcube, if any,

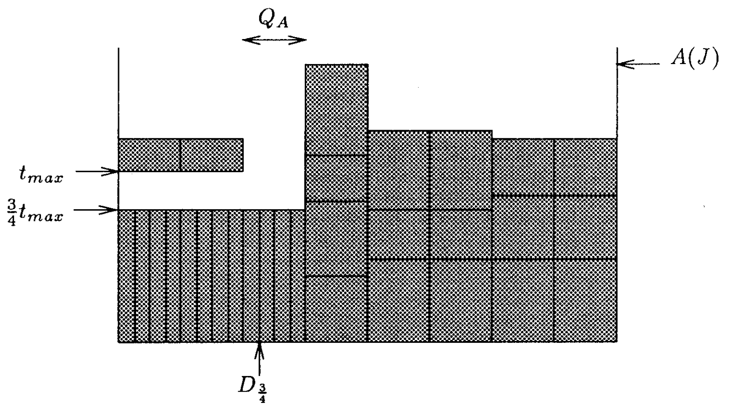

Figure 3: An example of the transformation

is selected; if J_j can be finished before t_{max} in the selected subcube, assign J_j to that subcube; otherwise, do nothing for J_j and consider the next job.
Step 4 Upon the assumption that every processors are available at t_{max}, apply LDF algorithm for the remaining jobs.

A sample schedule generated by Algorithm B is shown in Figure 4. Suppose that $J_l = (d_l, t_l)$ is the job finished at $B(J)$. Let Q_B be a d_l-subcube $[i2^{d_l}, (i+1)2^{d_l} - 1]$ such that $i2^{d_l} \leq D - 1 < (i+1)2^{d_l} - 1$. When D is a multiple of 2^{d_l}, no such a subcube exists. At that case, let $Q_B = \emptyset$.

Lemma 5 *In the schedule generated by algorithm B, no processor $p \notin Q_B$ is idle before $\frac{1}{2}t_{max}$.*

Proof. omitted. □
From Lemma 5, we can see that the number of processors that are idle before $\frac{1}{2}t_{max}$ is no more than 2^{d_l}. Since $D_{\frac{3}{4}} > 2^{m-1}$, the number of processors that are idle before $\frac{3}{4}t_{max}$ is also no more than 2^{m-1}. From these two observation, we can derive the following lower bound: $2^m h_{avg} \geq 2^m B(J) - (2^m - 2^{d_l})t_l - \frac{2^{m-1}}{4}t_{max} - \frac{2^{d_l}}{2}t_{max} - \frac{2^m}{4}t_{max}$ − − − **(3)**.
One additional observation is that $OPT(J) \geq 2t_l$. Now, it is easy to prove Lemma 6 from inequality (3).

Lemma 6 *For any set of jobs J with $D_{\frac{3}{4}} > 2^{m-1}$, $B(J) \leq (2 - \frac{1}{8})OPT(J)$.*

From Lemma 4, 6, we have Theorem 1. The time complexity $O(n \log n)$ in Theorem 1 can be achieved by using some typical implementation techniques, and the proof will be omitted.

Theorem 1 *There is an $O(n \log n)$ time scheduling algorithm with the absolute performance ratio no more than* 1.875.

3.3 Proof of Lemma 3

We need some additional notations. Let S be an arbitrary schedule. For any point of time t, let $ip_S(t)$ denote the set of processors that are idle at t, and, for any processor p, let $r_S(p)$ be the finish time of the last job assigned to p. For any job J_i, let $st_S(J_i)$ denote the time at which J_i starts to execute in S.

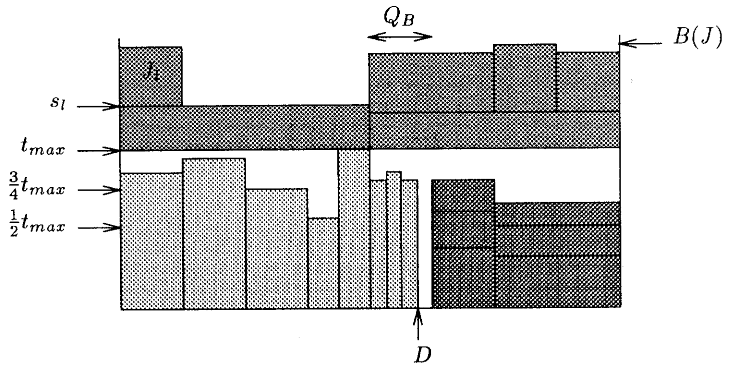

Figure 4: An example schedule generated by algorithm B

Lemma 7 *Let J be a set of n jobs with the dimension either d or 0. The jobs with the dimension zero have the same processing time t^0. For any schedule S (which may use non-basic subcubes) of J, there is another schedule S' such that*
(1) S' uses basic subcubes only;
(2) for any job J_i, $st_{S'}(J_i) \leq st_S(J_i)$;
(3) each d-subcube $Q_i = [i2^d, (i+1)2^d - 1], 0 \leq i \leq \frac{2^m}{2^d} - 1$, satisfies one of the following conditions:

(a) $\forall p, q \in Q_i, r_{S'}(p) = r_{S'}(q)$;

(b) Q_i is partitioned into two p-interval U_i and L_i such that $\forall p, q \in U_i, r_{S'}(p) = r_{S'}(q)$ and $\forall p', q' \in L_i, r_{S'}(p') = r_{S'}(q')$. Moreover, $r_{S'}(p) - r_{S'}(p') = t^0$, where $p \in U_i$ and $p' \in L_i$; in this case, Q_i is said to be a dirty subcube;

(4) moreover, there is at most one dirty subcube among $\frac{2^m}{2^d}$ subcubes.

Proof. Induction on n. When $n = 1$, it is trivial. Assume that the statements hold for any schedule of k jobs, and that S is a schedule of $k+1$ jobs. Let $J_{k+1} = (d_{k+1}, t_{k+1})$ be the job started lastly in S, and let S_k denote the schedule obtained by removing J_{k+1} from S. From the induction hypothesis, there is another schedule S'_k which satisfies the conditions. Note that, from condition (2), $|ip_{S'_k}(st_S(J_{k+1}))| \geq |ip_{S_k}(st_S(J_{k+1}))|$, and also that, from the condition (3), in the schedule S'_k, the idle processors at any point of time can be partitioned into a collection of basic d-subcubes and a p-interval with size less than 2^d.

Now, we have two cases. The first case is that $d_{k+1} = d$. In that case, select the earliest available basic d-subcube (which may be a dirty subcube), and assign J_{k+1} to that subcube. From the observations above, it is easy to verify that the resulting schedule satisfies the conditions. Now, examine the second case where $d_{k+1} = 0$. Suppose that there is a dirty subcube Q_i. Assign J_{k+1} to the first processor p of L_i at the time $r_{S'_k}(p)$. Note that $r_{S'_k}(p) \leq st_S(J_{k+1})$. If there is no dirty subcube, assign J_{k+1} to the first processor of the earliest available basic d-subcube. The resulting schedule also satisfies the conditions. □

Lemma 7 says that there is an optimal schedule in which only the basic subcubes are used. Moreover, the schedule generated by Lemma 7 also has the following property: if two jobs J_i and J_j with the dimension zero start at time t_i and t_j in the processors p and q, respectively, where p and q are in the same basic d-subcube, then either $t_i = t_j$ or $|t_i - t_j| \geq t^0$. With this property, we can prove Lemma 3.

Proof of Lemma 3

If the number of jobs with dimension zero is not a multiple of 2^d, there is a job J_i with the dimension zero in the optimal schedule such that J_i is assigned to a processor in a subcube Q_i, and there exists another processor in Q_i which is idle while J_i is running. We can add a new job $J' = (0, t^0)$ to that processor without increasing the overall finish time of the schedule. □

4 Conclusions

In this paper, we consider the problem of scheduling independent jobs in a hypercube system. In our setting, every job to be scheduled is initially available, and, moreover, the processing time of each job is also specified initially. However, there is a variety of different models for this problem. Several categories are commonly used in the scheduling theory to classify the problems.

- **Off-line vs. on-line** In off-line scheduling, every job is given at time 0, while, in on-line scheduling, each job is associated with an unknown ready time, and the existence of a job is unknown until the ready time of that job.
- **Clairvoyant vs. non-clairvoyant** [5] In clairvoyant system, the processing time of a job is given to the scheduler when the job is ready, while, in non-clairvoyant system, the processing time is not given until the execution of that job is completed.

Using these categories, let's discuss several variants of the problem considered in this paper.

Off-line, non-clairvoyant scheduling It is easy to show that no algorithm can do better, in the worst case, than LDF algorithm.

On-line, non-clairvoyant scheduling The technique given by [6] which converts an off-line algorithm into an on-line algorithm can be applied to this problem. If we use LDF algorithm as an off-line algorithm, we obtain an on-line algorithm with the absolute performance ratio 3.

Greedy rule is a natural strategy to schedule jobs in this environment. Formally speaking, a scheduling algorithm is said to be *greedy* if it does not leave any subcube idle so long as there is a job that can be executed in that subcube. Using Corollary 4.1 in [3], we can prove that no greedy algorithm can achieves an absolute performance ratio $3 - \epsilon$ for any constant $\epsilon > 0$.

References

[1] G. I. Chen and T. H Lai, "Scheduling independent jobs on partitionable hypercubes," *Journal of Parallel and Distributed Computing*, **12** (1991) pp. 74-78.

[2] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, **17** (1969) pp. 416-429.

[3] N. Graham, F. Harary, M. Livingston and Q. F. Stout, "Subcube fault-tolerance in hypercubes," *Information and Computation*, **102** (1993) pp. 280-314.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman Pub. Comp., San Francisco, 1979.

[5] R. Motwani, S. Phillips and E. Torng, "Non-clairvoyant scheduling," *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms* (1993) pp. 422-431.

[6] D. B. Shmoys, J. Wein and D. P. Williamson, "Scheduling parallel machines on-line," *Proc. 32nd Symposium on Foundations of Computer Science* (1991) pp. 131-140.

[7] Y. Zhu and M. Ahuja, "Job scheduling on a hypercubes," *IEEE Tr. Parallel and Distributed Systems*, **4** (1993) pp. 62-69.