An Efficient Approximation Algorithm for the File Redistribution Scheduling Problem in Fully Connected Networks

Ravi Varadarajan* Pedro I. Rivera-Vega†

Abstract

We consider the problem of transferring a set of files from their given locations in a fully connected network to their respective destinations in minimum time. The network has two unidirectional links, one in each direction, between every pair of nodes. We assume that it takes unit time to transfer a file across a link and no link is used by more than one file at any time. There is no restriction on storage capacity at the nodes. The objective of the File Redistribution Scheduling problem is to find routes for the files and a schedule for the use of the links along the routes so as to complete the transfer of all the files in minimum time. This problem has been shown to be **NP-hard** even with the restriction that each file must have at most one hop in its route. In this paper, we present an efficient polynomial time algorithm that finds an approximate solution to the problem. In this approximate solution, each file has at most one hop in its route and all the files can be transferred within *twice* the time taken by an optimal schedule.

1 Introduction

In this paper, we consider the problem of scheduling the redistribution of files in a computer network which is *fully connected*, that is, between any two nodes, there are two communication links, one in each direction. In this problem, the initial locations of files as well as their destinations are known beforehand. Every file takes constant

*Department of Computer and Information Sciences, University of Florida, Gainesville, FL 32611 ravi@cis.ufl.edu

†Department of Mathematics, University of Puerto Rico, Rio Piedras, P.R. 00931 p_rivera@upr1.upr.clu.edu

time (without loss of generality, assumed to be unit time) to move along any link. Each link can be used to transfer only one file during any time unit and every node in the network can simultaneously send and receive as many files as the number of outgoing and incoming links. A file can be moved in a *store and forward* fashion along a route from its source to its destination. There is no restriction on the storage capacity of the node. We also ignore the processing costs of the nodes in directing the transfer of files; this is a reasonable assumption since the complete schedule is determined beforehand and the nodes do negligible processing during the transfer. The objective is to find a schedule that minimizes the completion time of the transfer referred to as the *makespan* of the schedule. The schedule includes for each file, a route as well as the times for usage of the links in this route. In this paper, we do not address the issue of how the schedule is efficiently transmitted to the different nodes of the network.

The problem is of practical importance in distributed and parallel computer systems. For example, in a distributed database system, redistribution of data is periodically required in order to maintain a desired level of performance usually measured by average execution time of transactions[7]. In this case, the redistribution should be accomplished as soon as possible since the system cannot be fully operational while it is taking place. In the parallel processing context, this problem is related to the problem of routing packets among the processors of a network ([9, 5, 6]). In another related problem, referred to as *File Transfer Problem* (FTP), ([2, 3, 11]), the objective is to find a schedule with minimum makespan for the transfer of files in a network with no restriction in use of communication links but with the constraint that each file transfer requires communication modules at both ends; there is a limited number of such modules available at each node. Our problem is distinct from the FTP and to avoid confusion, we call our problem the "file redistribution scheduling problem".

The file redistribution scheduling problem has been shown to be intractable [8] even when the schedules are restricted to two-edge schedules. These are schedules in which every file has at most one hop in its route to its destination. In [8], the authors present a two-phase approach to find two-edge schedules. In the first phase, each file is assigned a two-edge route and in the second phase, a schedule for use of the links along the assigned routes is determined. The schedule determined in the second phase allows all the files to be transferred in minimum time for the pre-assigned routes. To determine the routes in the first phase, an IP formulation and a heuristic strategy were given. The former approach leads to an optimal two-edge schedule while the latter had been experimentally shown to provide close-to-optimal schedules.

In this paper, we present an algorithm which is guaranteed to find a schedule with

makespan that is no more than *twice* the makespan of an optimal schedule. In this approximate solution, each file has at most one hop in its route. Unlike the earlier approach, this algorithm finds routes as well as a schedule at the same time. We emphasize that in proving the accuracy bound of our approximate solution, we use a simple lower bound for the optimal makespan that in many cases may be close to *one half* the optimal makespan. This indicates that our approximation algorithm can provide an optimal solution in many (but not all) cases. The schedule determined by our algorithm has an advantageous characteristic in that at most n packets are queued up at a node during the file transfer (regardless of the number of packets involved in the transfer) where n is the number of nodes in the network. This can reduce the buffer capacity required at the intermediate nodes along a route.

This paper is organized as follows. In Section 2, we give a formal description of the problem with necessary notations. In Section 3, we present our approximation algorithm followed by its proof of correctness and time complexity analysis in Section 4. Finally, we summarize our results and discuss the scope of future work.

2 Problem Description and Notation

A *redistribution model* M is a tuple (V, E, S, D, F) where

$G = (V, E)$ is a directed graph, where V is the set of vertices representing the nodes of the network and E the set of edges representing unidirectional communication links. In our problem, $E = \{(u, v) | u, v \in V \text{ and } u \neq v\}$.

F - set of *files* that need to be reallocated, $F = \{f_1, f_2, \ldots, f_m\}$.

S - source function, $S : F \longmapsto V$ where $S(f_i)$ indicates the node where file f_i is initially located.

D - destination function, $D : F \longmapsto V$ where $D(f_i)$ indicates the node to which f_i has to be transferred.

It is assumed that $|V| = n$ and the nodes in G are numbered $1, 2, \ldots, n$. We say that a file f is of **type [i,j]** if $S(f) = i$ and $D(f) = j$. Any schedule is represented by a function $\mathcal{S} : E \times \mathcal{N} \rightarrow F \cup \{\epsilon\}$, where $\mathcal{N} = \{0, 1, 2, \ldots\}$; it is defined as follows:

$$\mathcal{S}(e, t) = \begin{cases} f & \text{if file } f \text{ uses link } e \text{ at time } t \\ \epsilon & \text{if no file uses link } e \text{ at } t \end{cases}$$

For each file f, let $t_f(\mathcal{S})$ be the time taken by f to move from its source to its destination in the schedule $\mathcal{S}$. The makespan of the schedule $\mathcal{S}$, denoted by $T(\mathcal{S})$, is

given by $\max_{f \in F} t_f(\mathcal{S})$. For a given redistribution model M, the **file redistribution scheduling problem** is one of determining a schedule with minimum makespan. We denote the optimum makespan for M by $T_{opt}(M)$.

In the case of fully connected networks, the transfer model is completely determined by an $n \times n$ non-negative integer matrix which we call the "requirement matrix". We will use M to denote also the requirement matrix corresponding to the transfer model M. In the requirement matrix M, $M[i,j] = m_{ij}$ where m_{ij} is the number of files of type $[i,j]$. Let $m_i = \sum_{j=1}^{n} m_{ij}$ denote the number of files having node i as the source, and $m_i^{'} = \sum_{j=1}^{n} m_{ji}$ the number of files having node i as the destination.

3 The Approximation Algorithm

The fact that the problem has been shown to be intractable [8] motivates us to seek efficient approximation strategies which guarantee schedules with bounded makespans with respect to optimal values. In this paper, we present one such approximation algorithm. Our algorithm constructs routes with at most one hop between sources and destinations. Before we present the details of the algorithm, we give the intuitive idea behind our approach. First we introduce some definitions and facts.

Definition 3.1 *The* **critical sum** *of an $n \times n$ requirement matrix M is given by $CS(M) = \max\left(\max_i \sum_{j=1}^{n} m_{ij}, \max_i \sum_{j=1}^{n} m_{ji}\right)$. A requirement matrix is* **perfect**, *if $\sum_j m_{ij} = \sum_j m_{ji} = CS(M)$ for all i, or in other words, the sum of entries of the matrix along any row or column is the same, equal to the critical sum of the matrix.*

Fact 3.2 *For an $n \times n$ requirement matrix M, let $M = \max_{1 \le i \le n} \{\max(\lceil \frac{m_i}{n-1} \rceil, \lceil \frac{m_i^{'}}{n-1} \rceil\}$. Then $T_{opt}(M) \ge L$, where $L = \lceil \frac{CS(M)}{n-1} \rceil$.*

Given an $n \times n$ requirement matrix M, let us define a bipartite multigraph $G_M = (V_M, E_M)$, where $V_M = \{u_i | 1 \le i \le n\} \bigcup \{v_j | 1 \le j \le n\}$ and E_M has m_{ij} edges between u_i and v_j for all $1 \le i, j \le n$. Suppose the requirement matrix M is an $n \times n$ *perfect matrix* with $CS(M) = n$. As we see later, this will allow us to decompose E_M into n complete matchings where a matching is a set of edges with no vertices in common and a complete matching includes all the vertices of the graph. Thus all the files that correspond to edges of a matching have no common source and common destination and if they are routed through the same intermediate node then they do not have conflicts in using the links. For example, in a four-node network, when a matching contains edges $(1,3)$ and $(2,4)$, we can route two files, one of type $[1,3]$ and another of type $[2,4]$ to go through an intermediate node (say) 4. This means that $[1,3]$ file will use the route $1 - 4 - 3$ and at the same time, the $[2,4]$ file will

use route $2-4-4$ meaning that it will use the direct link $(2,4)$ during the first time unit. Moreover, if we assign different intermediate nodes (note that there are at most n complete matchings) to routes of the files in different matchings, then all the files can be routed without any conflicts in using the links and the whole transfer can be completed in *two* time units. This schedule thus has optimal makespan for this case.

Extending this idea further, let us suppose that the $n \times n$ requirement matrix M is an arbitrary perfect matrix. Then we can decompose E_M into $CS(M)$ complete matchings, which can be divided into $\lceil \frac{CS(M)}{n} \rceil$ sets, each set containing n complete matchings with the possible exception of one set that can contain less than n matchings. We can now do routing in $\lceil \frac{CS(M)}{n} \rceil$ phases. In each phase, using the approach discussed above, we complete in 2 time units, the transfer of all the files that correspond to edges of the complete matchings contained in one set. Thus the whole transfer can be completed in $2\lceil \frac{CS(M)}{n} \rceil$ time units; by Fact 3.2, this is no more than $2L$ and hence no more than $2T_{opt}(M)$.

To tackle the case when M is an arbitrary $n \times n$ matrix, not necessarily perfect, we make use of the following result proved in [10].

Lemma 3.3 *Given any nonnegative integer square matrix M, there is a square matrix Z such that $M + Z$ is a perfect matrix and $CS(M + Z) = CS(M)$.*

The matrix Z simply indicates the "dummy" files that need to be transferred. The above result indicates that we can complete the transfer of all the files in M as well as in Z in $2T_{opt}(M)$. Note that a dummy file is not actually transferred but rather used as a theoretical device to denote "idle" states of links in a route. We are now ready to give the formal description of our algorithm.

Algorithm Schedule: Given M, the requirement matrix, produces a schedule $\mathcal{S}$.

begin

1. **FindPerfectMatrix**(M,E,W)
2. Construct a bipartite multigraph $G_W = (V_W, E_W)$ corresponding to the matrix W
3. $r := 0$
4. **while** $E_W \neq \emptyset$ **do**
   begin
5. **FindMaxMatching**(G_W,R)
6. $q := (r + 1) \bmod n$
7. $t := 2 * (r \text{ div } n)$
8. **while** $R \neq \emptyset$ **do**
   begin
9. Pick edge (u_i, v_j) in R

10. **if** $m_{ij} > 0$ **then**
begin
11. $\mathcal{S}((i,q),t) := f_{ij}$
12. $\mathcal{S}((q,j),t+1) := f_{ij}$
13. $m_{ij} := m_{ij} - 1$
14. Mark file f_{ij} as scheduled
end
15. $R := R - \{(u_i, v_j)\}$
end
16. $E_W := E_W - R$
17. $r := r + 1$
end
end Schedule

A few words on the algorithm. In step 5, we invoke the subroutine **FindMaxMatching** to find a maximum size matching R of the bipartite graph G_W. For this, we use the algorithm of [4]. In step 1, we invoke the subroutine **FindPerfectMatrix** to find a non-negative square matrix E and a perfect non-negative square matrix W with the same critical sum as M such that $M + E = W$. The proof of Lemma 3.3 given in [10] gives an implicit procedure to construct these matrices and we give an explicit efficient procedure below. The procedure we use is identical to the North-West-Corner (NWC) rule used in finding a feasible allocation in the transportation problem. In steps 11, 12, 13, and 14, f_{ij} represents any file of type $[i,j]$ that has not been marked as scheduled.

FindPerfectMatrix(M,E,W): Finds E and W such that $W+E = M$ and $CS(M) = CS(W)$.

begin
1. Compute $m_i := \sum_{j=1}^{n} m_{ij}$ and $m_i' := \sum_{j=1}^{n} m_{ji}$, $1 \le i \le n$
2. Initialize $E_{ij} := 0$ for all $1 \le i, j \le n$
3. Let $a_i = CS(M) - m_i$ and $b_j = CS(M) - m_j'$ for all $1 \le i, j \le n$
4. Find $k := \min\{1 \le i \le n | a_i > 0\}$ and $l := \min\{1 \le j \le n | b_j > 0\}$
5. If no such k or l exists then go to step 13
6. $E_{kl} := \min\{a_k, b_l\}$
7. $a_k := a_k - E_{kl}$ and $b_l := b_l - E_{kl}$
8. If $a_k = 0$ then set $k := \min\{k < i \le n | a_i > 0\}$
9. If no such k exists then go to step 13
10. If $b_l = 0$ then set $l := \min\{l < j \le n | b_j > 0\}$
11. If no such l exists then go to step 13

12. Go to step 6
13. $W_{ij} := E_{ij} + M_{ij}$ for all $1 \leq i, j \leq n$

end FindPerfectMatrix

4 Correctness and Makespan of the Schedule Produced

It is easy to see the termination of the proposed approximation algorithm. We need to show that the algorithm produces a valid schedule, and also estimate the deviation of the makespan of the schedule produced by the algorithm for a transfer model with respect to the the optimal makespan for the same transfer model. To prove this, we need the following result.

Lemma 4.1 *Let (X, E, Y) be a bipartite multi-graph in which the maximal degree of a node is p. Let X_1 be the set of nodes in X with degree equal to p. Then there exists a matching M which matches X_1 to a subset of Y.*

Proof: See [1], pp.46 ∎

Now we are ready to state the theorem that proves correctness of the algorithm.

Theorem 4.2 *The schedule $\mathcal{S}$ produced by the algorithm satisfies the following:*

(1) $\mathcal{S}$ is a valid schedule, and

(2) $T(\mathcal{S}) \leq 2T_{opt}(M)$.

Proof: We will assume the correctness of the two procedures **FindPerfectMatrix** and **FindMaxMatching**. To prove (1) we need to show that, under $\mathcal{S}$, each file is assigned to use a route from its source to its destination, and no two files are assigned to use the same link at the same time. Note that the iteration of step 4 of algorithm **Schedule** terminates only after all the files (corresponding to edges of the graph E_W) are assigned a two-edge route in steps 11 and 12. Suppose two files f_{ab} and f_{cd}, were both assigned to use the link (i, j), $i \neq j$, at the same time. Then either $i = a = c$ and the files are assigned routes $i - j - b$ and $i - j - d$ respectively or $j = b = d$ and the files are assigned routes $a - i - j$ and $c - i - j$ respectively. In both cases, they are assigned the same intermediate node and hence must have been part of the same maximum matching obtained in step 6. This is a contradiction since the matching has two edges incident on the same vertex.

To prove (2), we first show that in every iteration of the **while** loop of step 4, the maximum matching found in step 5 has size equal to n. By Lemma 4.1, it is sufficient

to show that the bipartite multigraph (V_W, E_W) that remains at the beginning of i-th iteration is a regular graph of degree $CS(M) - i + 1$. When $i = 1$, it is obvious since step (2) returns a perfect matrix with $CS(W) = CS(M)$. For the inductive step, we use the induction hypothesis and Lemma 4.1 to conclude that the claim is true for $i > 1$. Thus the **while** loop of step 4 is executed $CS(M)$ times and in each phase that consists of two time units, all the files (including dummy files) that correspond to edges in n complete matchings can be routed from their respective sources to their destinations. Thus the schedule has $\lceil \frac{CS(M)}{n} \rceil$ phases and has the makespan $t(\mathcal{S}) = 2\lceil \frac{CS(M)}{n} \rceil \leq 2L \leq 2T_{opt}(M)$. $\square$

Remark: The bound $2\lceil \frac{(n-1)L}{n} \rceil$ obtained above for the schedule produced by our algorithm is tight in the sense that it can be equal to optimal makespan. Consider an n node network, where $n > 3$. Let A be a subset of the vertex set with $|A| = \lfloor \frac{n}{2} \rfloor$. Suppose $(n-1)$ files need to be sent from each of the vertices in A to each vertex in $V - A$, where V is the vertex set of the graph. It is obvious that the optimal makespan is equal to $(n-1)$. On the other hand $L = \lceil \frac{n}{2} \rceil$. Thus the optimal makespan is equal to $\left(\frac{n-1}{\lceil n/2 \rceil}\right) L$, which is equal to the above bound when n is even and n divides L.

5 Time Complexity

Except for steps 1 and 5 in the algorithm, the remaining steps are straightforward, requiring only $O(1)$ time each. Step 1 invokes **FindPerfectMatrix** routine that is easily seen to have a time complexity of $O(n^2)$. For step 5, we can use the algorithm of [4]. For a bipartite graph $G = (V, E)$ it has worst-case running time $O((|V|+|E|)\sqrt{|V|})$. At the beginning of the i^{th} iteration of the **while** loop (step 4), the graph G_W has $|V| = 2n$ and $|E_W| = (CS(M)-i+1)n$. Suppose we define a new graph $G'_W = (V_W, E'_W)$ where $E'_W = \{(u_i, v_j) | \exists$ at least one edge from u_i to v_j in $E_W\}$. This new graph has at most one edge between two vertices. The number of edges of G'_W during iteration i is at most equal to $e_i = \min\{n^2, (CS(M) - i + 1)n\}$. Note that the graph G_W has a matching of size n if and only if the graph G'_W has a matching of size n. Thus the complexity of step 5 during iteration i is given by $O((n+e_i)\sqrt{n})$. Hence step 5 has overall worst-case running time $O(\sum_{i=1}^{CS(M)}(n+e_i)\sqrt{n})$ $= O(\min\{n^2 CS(M), (CS(M))^2 n\}\sqrt{n})$ which is equal to $O(n^{\frac{3}{2}} m \min\{m, n\})$ using the fact that $CS(M) = O(m)$, where m is the number of files. The overall complexity of the algorithm is thus seen to be $O(n^2 + n^{\frac{3}{2}} m \min\{m, n\})$.

6 Example

Consider a 4-node fully connected network with the requirement matrix M given as follows:

$$\begin{bmatrix} 0 & 2 & 3 & 2 \\ 5 & 0 & 2 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Thus $CS(M) = 7$. In step 1 of algorithm **Schedule**, we create the following perfect matrix W given below

$$\begin{bmatrix} 0 & 2 & 3 & 2 \\ 5 & 0 & 2 & 0 \\ 2 & 4 & 0 & 1 \\ 0 & 1 & 2 & 4 \end{bmatrix}$$

The **while** loop of step 4 is executed 7 times and hence the routing schedule is divided into $\lceil \frac{7}{4} \rceil = 2$ phases with each phase consisting of 2 time units. During each iteration, a $0-1$ permutation matrix is identified in step 5 of the algorithm and all the files corresponding to this matrix are routed through the same intermediate node. For example, in iterations 1-4, we can identify the following matrices.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Using the first matrix, we obtain routes $1-1-2, 2-1-1, 3-1-4$. Note that the first two files in the above routes use direct links $1-2$ and $2-1$ during the second and first time units respectively. Also the file of type $[4,3]$ is a "dummy" file and hence is not scheduled in steps 11 and 12 of the algorithm. Using the remaining three matrices, we obtain routes $1-2-3, 2-2-1, 1-3-3, 2-3-1$, $2-4-1$ and $1-4-4$ for the first two time units. For the second phase (iterations 5-7), we obtain the following permutation matrices.

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using these matrices, we obtain the routes $1-1-3, 2-1-1, 1-2-4, 2-2-3, 3-2-1$, $1-3-2, 2-3-3, 3-3-1$ for the third and fourth time units. The makespan is

4 time units for this example. Note that the optimal makespan for this example is 3 time units; to see this, let all the files except 2 files of type $[2,1]$ use direct link and let the two files of type $[2,1]$ follow routes $2-3-1$ and $2-4-1$ respectively.

7 Conclusions

In this paper, we have given an efficient algorithm for determining a schedule for the file redistribution scheduling problem in fully connected networks. This schedule has a makespan that is within twice the optimal makespan. Among many related open questions that we are currently investigating include the following: (1) Can we modify the proposed approximation algorithm so that it provides a schedule with makespan closer to optimal (especially when the optimal makespan is closer to L than to $2L$) ? and (2) Can we identify other cases, besides those discussed in [8], for which polynomial time algorithms to determine optimal schedules exist ? It is also possible to get improved schedules by first computing only the routes using the algorithm proposed in this paper and then determining the schedules using the scheduling algorithm proposed in [8]. The investigation of the accuracy of such schedules is also an interesting issue to explore.

References

[1] R. A. Brualdi, "Transversal Theory and Graphs," *Studies in Graph Theory, Vol.11, Part I*, 1975, pp.23-88.

[2] H.-A. Choi and S. L. Hakimi, "Data Transfers in Networks," *Algorithmica*, No. 3, 1988, pp. 223-245.

[3] E. G. Coffman, JR, M. R. Garey, D. S. Johnson and A. S. Lapaugh, "Scheduling File Transfers," *SIAM Journal on Computing* , Vol. 14, No. 3, August 1985, pp. 744-780.

[4] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, 2 (Dec. 1973),pp. 225-231.

[5] D. Peleg and E. Upfal, "The Generalized Packet Routing Problem," *Theoretical Computer Science*, 53, 1987, pp. 281-293.

[6] D. Peleg and E. Upfal, "The Token Distribution Problem," *SIAM Journal on Computing*, Vol. 18, No. 2, April 1989, pp. 229-143.

[7] P.I. Rivera-Vega, R. Varadarajan and S.B. Navathe, "Scheduling Data Redistribution in Distributed Databases", Technical Report, Database Research and Development Center, University of Florida, 1989.

[8] P.I. Rivera-Vega. R. Varadarajan and S.B. Navathe, "Scheduling Data Transfers in Fully Connected Networks," *Networks: An International Journal*, to appear.

[9] L. G. Valiant, "A Scheme for Fast Parallel Communication," *SIAM Journal on Computing*, Vol.11, 2, May 1982, pp. 350-361.

[10] E. A. Varvarigos and D. P. Bertsekas, "Communication Algorithms for Isotropic Tasks in Hypercubes and Wraparound Meshes," *Parallel Computing*, to appear.

[11] J. Whitehead, "The Complexity of File Transfer Scheduling with Forwarding," *SIAM Journal on Computing*, Vol. 19, No. 2, pp. 222-245, April 1990.