On a Local Protocol for Concurrent File Transfers

MohammadTaghi Hajiaghayi * †
Dep. of Computer Science
University of Maryland
College Park, MD
hajiagha@cs.umd.edu

Rohit Khandekar
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY
rohitk@us.ibm.com

Guy Kortsarz ‡
Dep. of Computer Science
Rutgers University-Camden
Camden, NJ
guyk@camden.rutgers.edu

Vahid Liaghat
Dep. of Computer Science
University of Maryland
College Park, MD
vliaghat@cs.umd.edu

ABSTRACT

We study a very natural *local* protocol for a file transfer problem. Consider a scenario where several files, which may have varied sizes and get created over a period of time, are to be transferred between pairs of hosts in a distributed environment. Our protocol assumes that while executing the file transfers, an individual host does not use any global knowledge; and simply subdivides its I/O resources *equally* among all the active file transfers at that host at any point in time. This protocol is motivated by its simplicity of use and its applications to scheduling map-reduce workloads.

Here we study the problem of deciding the start times of individual file transfers to optimize QoS metrics like average completion time or MakeSpan. To begin with, we show that these problems are NP-hard. We next argue that the ability of scheduling multiple concurrent file transfers at a host makes our protocol stronger than previously studied protocols that schedule a sequence of matchings, in which no two active file transfers share a host at any time. We then generalize the approach of Queyranne and Sviridenko (J. Scheduling, 2002) and Gandhi et al. (ACM T. Algorithms, 2008) that relates the MakeSpan and completion time objectives and present constant factor approximation algorithms.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Analysis of Algorithms and Problem ComplexityNon-numerical Algorithms and Problems; G.2.2 [**Mathematics of Computing**]: Discrete Mathematics—*graph theory, network problems*

General Terms

Theory

Keywords

local protocol, scheduling, file transfer, average completion time, MakeSpan

*Supported in part by NSF CAREER award 1053605 and Google Faculty Research Award.

†Part of this work was done while the authors were meeting at DIMACS. We would like to thank DIMACS for hospitality.

‡Supported in part by NSF grant number 0829959.

SPAA'11, June 4–6, 2011, San Jose, California, USA.

1. INTRODUCTION

1.1 Motivation

Today's technologies have enabled resources for compiling enormous amounts of data, often beyond the capacity of individual disks, and too large for processing with a single CPU. Such data is then naturally stored across a cluster of compute hosts and is shared and processed using distributed computing environments. One distributed computing paradigm which has attracted a lot of interest recently is Map-Reduce [25]. A typical Map-Reduce job has three phases. The *Map phase* processes data, often available on local disks, block-by-block and for each such block produces (key, value)-pairs that are written to the local disk. In a typical implementation, these pairs are organized into multiple files where each file corresponds to a specific key. The *Shuffle phase* transfers these files containing the (key, value)-pairs from the hosts that run Map tasks to the hosts that run Reduce tasks. Finally, the *Reduce phase* applies some function on all the values corresponding to each individual key and computes an output. The Shuffle phase involves the transfer of several files across multiple machines and can be quite I/O intensive. This phase can be naturally modeled as a file transfer problem. Let us denote the set of hosts participating in a Map-Reduce computation by V. The Map tasks may be present on two or more of these hosts. As a Map task finishes, the data produced by it becomes available for the transfer. Note that the amount of data produced by different Map tasks to be consumed by different Reduce tasks can be quite different. Let edge $e = \{u, v\}$ represent the data or a file to be transferred from a host u running a Map task to a host v running a Reduce task. Since we do not distinguish between the overheads caused by incoming or outgoing transfers, we model the transfers by undirected edges. Let $r(e)$ denote the time at which file e becomes available for transfer, namely its *release time*. Let $\ell(e)$ denote the size of file e. Although the exact values of $r(e)$ and $\ell(e)$ are not known a-priori, we assume that they can be estimated by some profiling mechanism. Once a file transfer has started, it cannot be preempted and it continues till completion. This assumption is important to eliminate the necessity to keep state and ensure fault tolerance. We next try to model the rates at which multiple active file transfers that share a host can proceed concurrently. The active file transfers originating or terminating at a common host share the disk and other I/O resources of that host. Thus if the number active file transfers at a host goes up, the rate of an individual file transfer goes down. To keep the model simple while capturing the essence of this resource sharing, we assume that if there are n active file transfers at a host, each file transfer can take place at a rate no more than $1/n$ times the rate on a dedicated host. Suppose at some point while file $e = \{u, v\}$ is being transferred, there are a total of n_u active file transfers at host u and a total of n_v active file transfers at host v. We assume that the effective rate at which e gets transferred is given by the minimum of the rates it can get at the two end hosts: $\min\{1/n_u, 1/n_v\}$. Of course, this rate may change over a period of time, since n_u or n_v may change with time. The scheduler for the Shuffle phase, thus has to decide the start times for the individual files. Once the start times are fixed, the files get transferred at rates given by the above model. There are several useful objective functions a scheduler may try to optimize. One such objective function is the average completion time. This MinSum objective gives an estimate on how soon the transfers finish so that the Reduce tasks can start. Another objective function is the MakeSpan. This MinMax objective captures the finish time of the last file transfer which, in turn, is a useful lower bound on the completion time of the overall Map-Reduce job.

1.2 Problem Formulation

A *file transfer model* is a triple (G, ℓ, r) where $G = (V, E)$ is an undirected multi-graph. The vertices V represent the hosts (or compute nodes) and the edges E denote the files to be transferred between the hosts. Assume that hosts are homogeneous and have identical processing capability of 1. For a vertex $v \in V$, let E_v denote the edges adjacent to v. The functions $\ell : E \mapsto \mathbb{N}$ and $r : E \mapsto \mathbb{N}$ denote the length and the release time of a file. The *uniform transfer model* is a transfer model where the length of all files are 1, i.e., $\forall e \in E, \ell(e) = 1$. The *zero-release transfer model* is a transfer model where all the files are available from the start, i.e., $\forall e \in E, r(e) = 0$. A schedule $\mathcal{S}$ defines a function $s_{\mathcal{S}} : E \mapsto \mathbb{R}^+ \cup \{0\}$, where $s_{\mathcal{S}}(e)$ is the starting time for the edge e. Once a file e is started to be transferred, it cannot be preempted and continues until it completes at time $f_{\mathcal{S}}(e)$. The rate at which the file is transferred however can vary over time and depends on how loaded the hosts u and v are at a particular time during its transfer. More precisely, for a time t, let $n_u(t)$ and $n_v(t)$ denote the total number of file transfers active at time t involving hosts u and v respectively. The effective processing e gets at time t is the minimum of the two processing capabilities at the end-points: $\min\{1/n_u(t), 1/n_v(t)\}$. This denotes the units of file transferred per unit time at time t. The transfer of file e continues till it gets a total processing of $\ell(e)$. There are two different cost functions which we like to minimize. In the *MakeSpan* version of the problem, the cost of a schedule $\mathcal{S}$ is the finishing time of the last transfer (job), i.e., $\text{MAX}(\mathcal{S}) = \max_{e \in E}\{f_{\mathcal{S}}(e)\}$. In the *MinSum* version, the cost of a schedule $\mathcal{S}$ is the sum of finishing times for all the file transfers, i.e., $\text{SUM}(\mathcal{S}) = \sum_{e \in E} f_{\mathcal{S}}(e)$. Note that the MinSum criterion corresponds to minimizing the average finishing time of all edges.

For any instance of the problem, consider opt_{ms} as a schedule with the minimum MakeSpan cost of $\text{OPT}_{ms} = \text{MAX}(opt_{ms})$. Similarly, consider opt_{sum} as a schedule with the MinSum cost of $\text{OPT}_{sum} = \text{SUM}(opt_{sum})$. We may omit the ms and sum indexes if they are clear from the context.

1.3 Our Model vs. the Non-Concurrent Model

The distinguishing feature of our model is that it allows multiple active file transfers at a host. If at each round only a matching can be scheduled, the model is called a non-concurrent model. Here we give simple examples in which our concurrent file transfer model gives better values of MakeSpan or average completion time by a constant factor than the non-concurrent model. Let $G = K_3$ be a triangle and let $r(e) = 0$ and $\ell(e) = 1$ for all $e \in G$. If we start transferring all three files at time 0, each file receives a rate of $1/2$ and completes at time 2, giving MakeSpan of 2. If however we insist that no two active files can share a host, the best way to schedule these edges is one-by-one, giving a MakeSpan of 3. Moreover, the MakeSpan problem with uniform file sizes has a simple scheduling in the concurrent file transfer (see Section 4), in the Non-concurrent model it is equivalent to properly coloring the edges of a graph, which is an NP-Complete problem (see for example [8]). We can also give a simple example for the MinSum criteria. Let G be a path of length 2 with edges e and e'. Let $\ell(e) = M$, $\ell(e') = 1$, $r(e) = 0$ and $r(e') = M/2$. If we start transferring files right on their release times, the sum of completion times would be $f(e) + f(e') = (M + 1) + (M/2 + 2) = 1.5M + 3$. But if we are not able to transfer the files incident to a host concurrently we must transfer them in some order. If we start transferring e at time zero, then the total cost would be $M + (M + 1) = 2M + 1$. If we want to transfer e' before e, then the total cost would be $(M/2 + 1) + (M/2 + 1 + M) = 2M + 3$. Therefore the cost of the optimum solution is almost $2M$ in the Non-concurrent model but the cost of the optimum in concurrent model is almost $1.5M$.

In this paper, we design our algorithms by mostly relying on non-concurrent scheduling algorithms, however, the approximation ratio is defined by comparing the cost of an algorithm with that of an optimal concurrent solution. Note that an algorithm with an approximation ratio α in our model has at most the same approximation ratio compared against an optimal non-concurrent solution. However, the reverse is not true: a solution which approximates the optimal non-concurrent solution within a factor β, is not necessary a β-approximation solution in our model.

1.4 Our Contributions

THEOREM 1. *The problem of computing a schedule with minimum MakeSpan is NP-complete. The problem of computing that of minimum average completion time is NP-complete even on trees, and even with file sizes* 1 *or* 2.

In Sections 3.2 and 3.1, we then present constant factor approximations for our problems.

THEOREM 2. *There exist polynomial-time algorithms that achieve the following approximation factors for various versions of the problems we study*[1]. *Here* $\mathrm{e} \approx 2.718$ *stands for the base of the natural logarithms.*

[1]The approximation ratio for the Avg. Resp. Time on General Non-Uniform version is claimed to be $6\mathrm{e}$ in the conference version. However, due to a subtle flaw it is corrected to $9\mathrm{e}$.

File sizes	*Release Times*	*MakeSpan*	*Avg. Resp. Time*
Non-Uniform	*General*	3 *§3.2*	$9e$ *§3.3*
	Zero-Release	2 *§3.2*	$4e$ *§3.3*
Uniform	*General*	3 *§3.2*	$6e$ *§3.3*
	Zero-Release	1 *§4*	3.658 *§4*
			Bipartite: $\sqrt{2}$ *§4*

Our techniques. We generalize the technique given by Queyranne and Sviridenko [18] and Kortsarz et al. [3] for our framework in section 3.1. They use a method to reduce the MinSum criteria to MakeSpan criteria. Their basic idea is to first partition the vertices into disjoint subsets according to a certain node-weight function. Next, they reduce the MinSum problem on every subset to a MakeSpan problem induced by that subset. However, their analysis strongly relies on specific properties of both the partitioning function and the particular MakeSpan algorithm used in their solution. We generalize this technique to partition the edges (instead of vertices) using an arbitrary partitioning function and then using a scheduling algorithm with very simple restrictions. This Meta-algorithm provides a general tool for scheduling conflicting jobs under the MinSum criteria. Using this approach, we present constant approximation algorithms for the general file transfer problem with the MinSum criteria.

1.5 Related Work

A closely related problem to the file transfer problem is the "Data Migration" problem. The data migration problem arises in large storage systems, such as Storage allocation or Scheduling on dedicated processors [3], where a network of hosts is used to store multimedia data. As the data access pattern changes over time, the load across the hosts needs to be re-balanced. This is done by computing a new data layout and then "migrating" data to convert the initial data layout to the target data layout. The migration is done by transferring files between the hosts. A host completes when all the files concerning that host is transferred. Clearly it is important to compute a data migration schedule that converts the initial layout to the target layout quickly.

This problem can be modeled as a transfer graph, in which the vertices represent the hosts and an edge between two vertices u and v corresponds to a data object that must be transferred from u to v, or vice-versa. Each edge has a length that represents the transfer time of a data object between the hosts corresponding to the endpoints of the edge. In data migration problem we assume that any host can be involved in at most one transfer at any time. In contrast to the file transfer problem, in the data migration problem the goal is to minimize the completion time of the hosts (vertices) instead of that of files (edges). Several variations of the data migration problem have been studied, arising either due to different objective functions or due to additional constraints.

There are usually three different objective functions to optimize. One common objective function is to minimize the MakeSpan of the migration schedule, i.e., the time by which all migrations complete. Coffman et al. [9] introduced this problem. They showed that when edges may have arbitrary lengths, a class of greedy algorithms yields a 2-approximation to the minimum MakeSpan. In the special case where the lengths are uniform, i.e., edges have equal (unit) lengths, the problem reduces to edge coloring of the transfer (multi)graph of the system for which an asymptotic approximation scheme is now known [13].

Another objective function is to minimize the average completion time over all hosts (vertices). In this version we usually consider the weighted sum of finishing times of vertices, i.e., each vertex v is associated with a weight $w(v)$ and the completion time of each vertex $C(v)$ is the last finishing time of edges adjacent to v and we want to minimize $\sum_v w(v)C(v)$. Kim [6] proved that the problem is NP-hard and showed that Graham's list scheduling algorithm [14], when guided by an optimal solution to a linear programming relaxation, gives an approximation ratio of 3. She also gave a 9-approximation algorithm for the case where edges have arbitrary lengths. Gandhi et al. [11] showed that the analysis of the 3-approximation algorithm is tight. They also gave a 5.06-approximation algorithm for a more general case when edges have release times and arbitrary lengths.

Bar-Noy et al. [10] studied the data migration problem with the objective to minimize the average completion time over all data migrations (edges). They showed that the problem is NP-hard and gave a simple 2-approximation algorithm for the uniform case (which is also known as *Min Sum Edge Coloring Problem*). Halldórsson et al. [2] improved this ratio to 1.8298. For arbitrary edge lengths, the best known ratio is 7.682 by [3].

A problem related to the data migration problem is open shop scheduling. In this problem, we have a set of machines and a set of jobs with positive weights. Each job consists of a set of operations which can be performed in any order. Each operation has a processing time and must be processed on a specific machine. Each machine can process a single operation at any time, and two operations that belong to the same job cannot be processed simultaneously. The objective is to minimize the sum of weighted completion times of all jobs. This problem is a special case of the data migration problem [11]. Open shop scheduling problem has been studied in [11, 15, 16, 17, 18].

For different models of data migration, see [21, 20, 19].

2. NP-COMPLETENESS RESULTS

NP-Completeness for min. avg. completion time.

THEOREM 3. *The problem of computing a schedule with minimum average completion time is NP-hard even in trees and even if the jobs have length* 1 *or* 2.

The reduction uses the ideas of Marx [22]. The problem Marx considered is *preemptive sum multicoloring of the edges of the a tree* (MEPS). In this problem we are given a tree and every edge has an integral length $\ell(e)$. We have to color the edges of the tree with positive integers $1, 2, 3, \cdots$. If e and e' share a vertex, then their color sets must be disjoint. Thus a solution must choose a matching at every round (and the edges in the matching get the color of the round number). Every edge e must belong to $\ell(e)$ matchings. Let Ψ be a proper coloring and $f_\Psi(e)$ be the largest color assigned to e by Ψ. The goal is to minimize $\sum_{e\in E} f_\Psi(e)$. In the non-preemptive case, every e must receive $\ell(e)$ *consecutive* integers. Consider MEPS. The solution of [22] for a YES instance happens to be *non-preemptive* as we shall see. Note that this implies a hardness for the non preemptive case as well. For us this fact is important as it fits our model which is non-preemptive. We now state some observations used by [22] (albeit, not made explicitly in [22]).

Definition Given an undirected graph $G(V, E)$ a *vertex cover* (of the edges) is a subset $A \subseteq V$ so that for every edge $e = uv \in E$ either $u \in A$ or $v \in A$. An *exact vertex cover* A, is a vertex cover A, so that for every $e = uv$ *exactly one of* u or v belongs to A.

Consider an exact vertex cover A of the edges of the graph. Say that $v, u \in A$. Let E_v, E_u be the edges of v and u in G.

CLAIM 1. $E_v \cap E_u = \emptyset$.

Proof. If $E_v \cap E_u \neq \emptyset$ then $E_v \cap E_u = uv$ as uv is the only edge that can appear both in E_v and in E_u. But our assumption that $u \in A$ and $v \in A$ implies that A is not an exact cover (uv is covered twice). This is a contradiction. □

Hence the collection of sets of edges $\{E_v \mid v \in A\}$ is a collection of *edge-disjoint stars*, with v the center of E_v. Note that every edge appears in exactly one of these stars.

Definition A perfect coloring of a star E_v of A is a coloring that takes the star, deletes the rest of the edges from the graph, and assigns this star its optimum coloring (disregarding conflicts that may occur with other stars). A perfect coloring, is a perfect coloring of all stars $\{E_v, v \in A\}$.

It is not clear a-priori that a perfect coloring exists. However, we can prove the following.

CLAIM 2. *If we can find an exact cover A and it is possible to find perfect and proper [2] coloring of all stars, then the coloring is optimal.*

Proof. Since the stars are edge-disjoint we consider every star separately. Given a star collection of A, the perfect sum coloring corresponding to each star lower bounds the contribution of this star to the sum. This is because this coloring is locally optimal (disregarding all other stars). Thus the sum of the contribution of a perfect coloring over all stars of A is a lower bound on the optimum sum. Since we assume that a perfect coloring can be obtained in a consistent way, the coloring is optimal. □

Consider a general star with center v and let $E_v = e_1, e_2, \ldots, e_k$. Assume without loss of generality that $\ell(e_1) \leq \ell(e_2) \leq \ldots \leq \ell(e_k)$.

CLAIM 3. *In the MinSum problem, a perfect coloring of the star first schedules e_1 for $\ell(e_1)$ time units, and then schedules e_2 for $\ell(e_2)$ time units, and so on.*

Proof. Let $i < j$. The edges e_i and e_j share a vertex. This means that one of the edges will add to the delay of the other or vice versa. Since $\ell(e_i) \leq \ell(e_j)$, the contribution to the delay of this pair is at least $\min\{\ell(e_i), \ell(e_j)\} = \ell(e_i)$. For this e_i has to be transferred fully before the transfer of e_j starts. The option of scheduling edges together, which holds in our file transfer model, does not produce a perfect coloring. It works in 'half speed'. More precisely, if we schedule e_i, e_j for $\epsilon > 0$ time units together, then e_i and e_j both get ϵ units of delay and $\epsilon/2$ units of their jobs were finished by now. Now, even if the $\epsilon/2$ time units of the mutual schedule were the last time units e_i needed, the rest of the job e_i delayed e_j by $\ell(e_i) - \epsilon/2$. Thus the total delay corresponding to these two edges is $\epsilon + (\ell(e_i) - \epsilon/2) = \ell(e_i) + \epsilon/2$ which is a non-perfect coloring. It is easy to see that after e_i and e_j were scheduled for $\epsilon > 0$ time units together, *no coloring completion* can derive a perfect coloring. Now, if we schedule e_1 fully first and then e_2 and so own, for every $i < j$ the delay caused by the pair e_i to e_j is the minimum possible $\ell(e_i)$. Hence all delays for all edge pairs is the minimum possible and the coloring is perfect. □

The following theorem is proved in [22]. We start with a 3−SAT instance so that every literal appears twice non-negated and twice negated. This problem is still NP-hard (see [24]). We denote this problem by 3−SAT−4. In [22] a reduction from 3−SAT−4 to MEPS is given. The instance of MEPS is called a YES *instance* if it corresponds to a satisfiable 3−SAT−4 formula. Else it is called a NO instance.

[2] A coloring is proper if no two edges of the same color share an endpoint.

THEOREM 4. [22] *In the* YES *instance of* MEPS*, there exists an exact vertex cover A and a perfect proper coloring of its stars. In addition, the coloring is* non-preemptive. *A* NO *instance, does not admit a perfect proper coloring of A and so the sum is larger than the one given by a perfect sum.*

In [22] it is shown that there exist an optimum solution for a MEPS instance such that the maximum color is at most $p \cdot (2\Delta - 1)$ with p the maximum demand and Δ the maximum degree in the graph. If p is exponential in n then a solution for MEPS is exponential (in the NO instance, as the preemptive coloring is concerned, there may be no short description of the coloring). Therefore we assume that p is bounded by a polynomial in n.

COROLLARY 1. *If there exists a polynomial time algorithm for the MinSum problem, then $P = NP$.*

Proof. By Claim 3, one can compute the cost of a perfect coloring in polynomial time. Thus it is sufficient to show an instance is a YES instance if and only if OPT_{ms} is equal to the cost of a perfect coloring.

Since the solution of [22] for a YES instance is non-preemptive, the solutions of MEPS and the MinSum problem are *identical* for a YES instance. Hence by Theorem 4, in a YES instance the optimum solution corresponds to a perfect coloring. On the other hand, in a NO instance the optimum (preemptive) solution for MEPS and the optimum solution for the MinSum problem have nothing in common. However, by Claim 2 the value of the NO instance for both models is larger than the value of a perfect coloring. Therefore given a polynomial algorithm for the MinSum problem, we can distinguish between a YES instance and a NO instance of the 3−SAT−4 problem. □

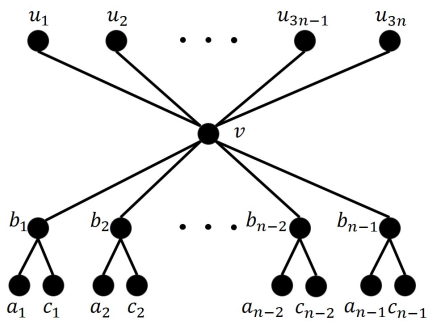
NP-Completeness for MakeSpan.

We reduce the strongly NP-hard problem called *3-partition* to the problem of computing a schedule with minimum MakeSpan. An instance of the 3-partition problem is given by an integer $B > 0$ and $3n$ integers $s_1, \ldots, s_{3n}$ such that $B/4 < s_i < B/2$ for all $i \in [3n]$ and $\sum_{i=1}^{3n} s_i = nB$. Here $[k]$ stands for $\{1, \ldots, k\}$. An instance is called a YES instance if the $3n$ integers can be partitioned into subsets $G_1, \ldots, G_n$ such that each G_j has 3 elements adding up to exactly B. An instance that is not a YES instance is called a NO instance. It is well known that it is strongly NP-hard to distinguish between YES and NO instances [24].

We now give a polynomial-time procedure that, given an instance of the 3-partition problem, creates an instance of the MakeSpan minimization problem. The graph G of the MakeSpan minimization instance created is given in Figure 1. The release times and lengths of the various edges are also given in the adjacent table.

LEMMA 1. *An instance obtained from a* YES *instance of the 3-partition problem has optimum MakeSpan $(B+1)n - 1$, while an instance obtained from a* NO *instance of the 3-partition problem has optimum MakeSpan strictly more than $(B+1)n - 1$.*

Proof. Consider an instance obtained from a YES instance of the 3-partition problem. We create a non-concurrent schedule (i.e., a schedule which does not schedule multiple edges incident to a host at any point in time) with MakeSpan $(B+1)n - 1$. First schedule all the edges of the form $\{v, b_j\}, \{b_j, a_j\}, \{b_j, c_j\}$ for $j \in [n-1]$ starting at their respective release times. Note that no two adjacent edges among them will be active at any point in time. Now the only time windows that are open for scheduling edges of the form



(a)

Edge e	Release time $r(e)$	Length $\ell(e)$
$\{v, u_i\}$ for $i \in [3n]$	0	s_i
$\{v, b_j\}$ for $j \in [n-1]$	$(B+1)j-1$	1
$\{b_j, a_j\}$ for $j \in [n-1]$	0	$(B+1)j-1$
$\{b_j, c_j\}$ for $j \in [n-1]$	$(B+1)j$	$(B+1)(n-j)-1$

(b)

Figure 1: MakeSpan minimization instance in the reduction

$\{v, u_i\}$ for $i \in [3n]$ are $W_j = [(B+1)(j-1), (B+1)(j-1)+B)$ for $j \in [n]$. Note that each of these windows is of length exactly B. Let $G_1, \ldots, G_n$ be the partition of the $3n$ integers into subsets of size 3 each adding up to B. Fix $j \in [n]$ and let $G_j = \{s_{j_1}, s_{j_2}, s_{j_3}\}$ so that $s_{j_1} + s_{j_2} + s_{j_3} = B$. Now schedule the edges $\{v, u_{j_1}\}, \{v, u_{j_2}\}, \{v, u_{j_3}\}$ in window W_j one after the other. This in fact gives a non-concurrent schedule with MakeSpan $(B+1)n-1$.

Now it is enough to show that if there is a schedule (either concurrent or non-concurrent) with MakeSpan at most $(B+1)n-1$, the instance must have been created from a YES instance of the 3-partition problem. Note that the edges incident to each b_j have total length exactly $(B+1)n-1$. Considering their release times, it is clear that for the MakeSpan to be at most $(B+1)n-1$, these edges must be schedules at their respective release times in a non-concurrent manner. This again gives that the only time windows that are open for scheduling edges of the form $\{v, u_i\}$ for $i \in [3n]$ are W_j for $j \in [n]$ defined above. Furthermore no edge $\{v, u_i\}$ can be active in more than one window. Thus each edge $\{v, u_i\}$ maps to a unique window W_j. The edges mapping to any single window must have total length exactly B. This naturally induces a feasible solution to the 3-partition problem and hence gives that the starting instance of the 3-partition problem must have been a YES instance. □

3. NON-UNIFORM TRANSFER MODEL

In this section we present different algorithms for finding schedules under the non-uniform transfer model. In Section 3.1, we generalize the approach in [4] and [3] to give a meta-algorithm for solving the MinSum problem by partitioning the jobs (files) into different blocks and then using an algorithm with good MakeSpan time for each block.

In Section 3.2, we give constant competitive algorithms for the MakeSpan problem. Finally in Section 3.3, by providing different bucketing functions and using algorithms in Section 3.2, we give constant competitive algorithms for the MinSum problem.

3.1 Overview of the Approach for Solving the MinSum Problem

We use a meta-algorithm which provides a very general tool for scheduling conflicting jobs under the minimum sum criteria. The meta-algorithm uses a *bucketing function* f^* to divide the jobs into disjoint blocks such that each block has a "uniformity property" (e.g., in a near optimum scheduling these blocks end up roughly having the same finishing time). Then we simply schedule each block using a MakeSpan algorithm $\mathcal{A}$. The trick is to find a bucketing function such that the sum of values assigned to jobs is in a small constant approximation of OPT_{sum} (this imposes an upper bound on *bucket values*), and the maximum bucket value should also be a constant approximation of OPT_{max} (this imposes a lower bound on bucket values).

Now we elaborate how to partition the job set E into different blocks $E^0, E^1, \ldots, E^k$. For an instance (G, ℓ, r), assume a value $f^*(e)$ (which we call the *bucket value* of e) is associated with each job e. Let $a > 1$ be a constant real number and α be a value chosen uniformly at random from $[0, 1)$. Let $l_i = a^{\alpha+i}$, for $i = -1, 0, 1, \ldots, k$. Define the block $E^i = \{e \in E | l_{i-1} < f^*(e) \leq l_i\}$, for $i = 0, \ldots, k$. So the edge set E, is divided into disjoint blocks of $E^0, \ldots, E^k$. Denote by b_e the block into which edge e belongs (which of course, is a function of α). The meta-algorithm $\mathbf{ALG}(\mathcal{A}, f^*)$ given in the figure, applies $\mathcal{A}$ (which will give us a near optimum MakeSpan time) on all the blocks separately. We may use the same notation $\mathbf{ALG}(\mathcal{A}, f^*)$ as the schedule given by this algorithm on an instance of the MinSum problem.

Algorithm 1 $\mathbf{ALG}(\mathcal{A}, f^*)$

Choose α uniformly at random from $[0, 1)$.
Using the bucket function f^*, partition the edges into blocks $E^0, \ldots, E^k$.
Schedule the blocks in sequence using algorithm $\mathcal{A}$.

We give sufficient properties for the bucket function (regarding the MakeSpan algorithm $\mathcal{A}$ and the optimum answer OPT_{sum}) in order for $\mathbf{ALG}$ to have a constant approximation ratio. Let f^* be a bucketing function and $\mathcal{A}$ be a scheduling algorithm such that for any instance of the transfer problem $\langle G, \ell, r \rangle$:

(P1) For any subgraph $H = (V', E') \subseteq G$ we have $\forall_{e \in E'} \exists_{e' \in E'} f_{\mathcal{A}}(e) \leq \beta f^*(e')$ where $f_{\mathcal{A}}(e)$ is the finishing time of e if we run A only on H.

(P2) $\sum_{e \in E} f^*(e) \leq \gamma \text{OPT}_{sum}$

where β and γ are constants. Lemma 2 gives an upper-bound on the expected value of finishing times of edges in $\mathbf{ALG}$.

LEMMA 2. *Let f^* and $\mathcal{A}$ have the property (P1) and let $\mathbf{ALG}(\mathcal{A}, f^*)$ be the corresponding schedule. For each edge $e \in E$, $\mathbf{E}[f_{\mathbf{ALG}}(e)] \leq \beta \frac{a}{\ln a} f^*(e) \leq \beta e f^*(e)$. Here $e \approx 2.718$ stands for the base of the natural logarithms.*

Proof. In $\mathbf{ALG}$ before scheduling the block i, we wait for blocks $0 \leq j < i$ to finish transferring all their files, and then we schedule the edges in E^i, using $\mathcal{A}$. To bound the finishing time of edges we consider an arbitrary block i separately and then we consider the waiting time required for transferring previous blocks.

Let $f_{\mathcal{A}}(e)$ be the finishing time of edge e when the block E^{b_e} has been scheduled separately by $\mathcal{A}$. According to (P1) for $G = \langle V, E^{b_e} \rangle$ we have:

$$f_{\mathcal{A}}(e) \leq \beta f^*(e') \leq \beta l_{b_{e'}} = \beta l_{b_e}$$

in which the last equality follows from the fact that e and e' are in the same block. This shows that if we consider each block i

separately, an edge e finishes at most on βl_{b_e}. Recall that $l_i = a^{\alpha+i}$. Considering the waiting time for blocks before b_e, we have:

$$f_{\mathbf{ALG}}(e) \le \sum_{i=0}^{b_e} \beta l_i = \beta \sum_{i=0}^{b_e} a^{\alpha+i} \le \frac{\beta a^{\alpha+b_e+1}}{a-1} = \beta \frac{a}{a-1} t_{b_e}$$

where $t_{b_e} = a^{\alpha+b_e}$. Since α is a random variable, then b_e and t_{b_e} are also random variables. We use the same method as in [3] to compute the expected value of t_{b_e}. Let $z = \log_a f^*(e)$, for $e \in E$. Define $y_e = \alpha + b_e - z$. Since b_e is the smallest integer such that $\alpha + b_e \ge z$, then y_e is uniformly distributed on $[0, 1)$. Therefore

$$\begin{aligned} \mathbf{E}[f_{\mathbf{ALG}}(e)] &\le \beta \frac{a}{a-1} \mathbf{E}[a^{y_e+z}] = \beta \frac{a}{a-1} f^*(e) \int_0^1 a^t dt \\ &= \beta \frac{a}{a-1} f^*(e) \frac{a-1}{\ln a} = \beta \frac{a}{\ln a} f^*(e). \end{aligned}$$

The function $f(a) = \frac{a}{\ln a}$ is maximized when $a = \mathrm{e} \approx 2.718$, therefore $\mathbf{E}[f_{\mathbf{ALG}}(e)] \le \beta \mathrm{e} f^*(e)$. □

Considering Lemma 2 we would have the following.

THEOREM 5. *The sum cost of* $\mathbf{ALG}(\mathcal{A}, f^*)$ *is less than* $\beta\gamma\mathrm{e}\text{OPT}_{sum}$ *for any instance of transfer problem* (G, ℓ, r) *and bucketing function* f^* *and scheduling algorithm* $\mathcal{A}$ *with both properties (P1) and (P2).*

Proof. By Lemma 2, $\text{SUM}(\mathbf{ALG}) = \sum_{e\in E} f_{\mathbf{ALG}}(e) \le \beta\mathrm{e} \sum_{e\in E} f^*(e)$. Thus by (P2) we have $\text{SUM}(\mathbf{ALG}) \le \beta\gamma\mathrm{e}\text{OPT}$. □

3.2 MakeSpan Problem

We call a schedule $\mathcal{S}$, *non-concurrent schedule* if by applying that schedule no processor runs more than one transfer at any time, i.e., for any two adjacent edges e and e', we have either $s(e) \ge f(e')$ or $s(e') \ge f(e)$. Let deg_v for any $v \in V$ denote the degree of v in G. Next theorem shows that a greedy algorithm can guarantee a 3-factor of the optimum solution in the MakeSpan version.

THEOREM 6. *There is an algorithm* Greedy MakeSpan *(or* $\mathbf{GMS}$*) which for any instance of transfer problem* (G, ℓ, r) *gives a non-concurrent schedule with a MakeSpan cost at most* 3OPT_{ms}*. In addition for every edge* $e = \{u, v\} \in E$ *the finishing time is at most* $r(e) + \sum_{e'\in E_u} \ell(e') + \sum_{e'\in E_v} \ell(e') - \ell(e)$.

Proof. We use a sweep line method to make a non-concurrent schedule (thus $f(e)$ would be equal to $s(e) + \ell(e)$ for any edge $e \in E$). In each step of the algorithm given in the figure, we schedule the edge which can start sooner than all other edges which are not yet scheduled, breaking ties arbitrarily. Formally, consider U_i as the subset of edges which are scheduled by our algorithm before step i. Initially U_1 is empty and at each step of the algorithm we schedule a new edge. Recall that E_v for any $v \in V$, denotes the edges adjacent to v. In step i, for any edge $e = \{u, v\} \notin U_i$, consider $p_i(e)$ as the first *possible* starting time of e, i.e., the time after its release time and after transferring all currently scheduled edges which are adjacent to u and v. Let e_i be the edge with minimum possible starting time in step i. We set the starting time of e_i equal to $p_i(e)$, set $U_{i+1} = U_i \cup \{e_i\}$, and continue the algorithm in the next step. We observe that the resulting sequence of starting times $s(e_1), \ldots, s(e_{|E|})$ is non-decreasing.

Algorithm 2 Greedy MakeSpan ($\mathbf{GMS}$)

Input: An instance of transfer problem $(G = (V, E), \ell, r)$.
Output: A schedule $\mathcal{S}$ with corresponding starting times for each edge in E.

Define $U_1 = \phi$.
for $i = 1$ to $|E|$ **do**
  For every edge $e' = \{u, v\} \notin U_i$, define $p_i(e) = \max\{r(e), \max_{e'\in U_i\cap(E_u\cup E_v)} f_{\mathcal{S}}(e')\}$.
  Let $e_i \in \text{argmin}_{e\notin U_i} p_i(e)$.
  Set $s_{\mathcal{S}}(e_i) = p_i(e_i)$ and thus $f_{\mathcal{S}}(e_i) = s_{\mathcal{S}}(e_i) + \ell(e_i)$.
  Define $U_{i+1} = U_i \cup \{e_i\}$.

Now we show that for each edge $e^* = \{u, v\}$, $s(e^*)$ is at most $r(e^*) + \sum_{e\in E_u} \ell(e) - \ell(e^*) + \sum_{e\in E_v} \ell(e) - \ell(e^*)$. To show this, we will prove by contradiction that u and v are never idle in the same time between $r(e^*)$ and $s(e^*)$. Let $r(e^*) \le t < s(e^*)$ be the moment that both u and v are idle. Let k be the first step which we schedule an edge after t, i.e., $k = \min_{\{i|s(e_i)>t\}} i$. In step k, $p_k(e_k) > t$ and $s(e) \le t$ for all edges $e \in U_k$. Since u and v are both idle on time t and no scheduled edge is started after that time, the minimum possible starting time of e^* at step k is $p_k(e^*) = t$. But the edge with minimum possible starting time at step k is e_k with $p_k(e_k) > t$, which is a contradiction.

Therefore the finishing time of each edge $e^* = \{u, v\}$ is at most

$$\begin{aligned} f(e^*) &= s(e^*) + \ell(e^*) \\ &\le r(e^*) + \sum_{e\in E_u} \ell(e) + \sum_{e\in E_v} \ell(e) - \ell(e^*) \\ &\le 3\text{OPT} - \ell(e^*). \end{aligned}$$

The last inequality follows from the fact that $\text{OPT} \ge \sum_{e\in E_v} \ell(e)$ for any vertex $v \in V$ and $\text{OPT} \ge r(e) + \ell(e)$ for any edge $e \in E$. □

We note that by Theorem 6, any edge $e = \{u, v\}$ with release time of zero finishes at most on $\sum_{e'\in E_u} \ell(e') + \sum_{e'\in E_v} \ell(e') - \ell(e) \le 2\text{OPT}$. Hence in a zero-release transfer problem $\mathbf{GMS}$ is always in 2-approximation of the optimum.

COROLLARY 2. *For any instance of zero-release transfer problem* (G, ℓ)*,* $\mathbf{GMS}$ *gives a non-concurrent schedule with MakeSpan cost at most* 2OPT_{ms}.

3.3 MinSum Problem

In this section we present a bucketing function with both properties (P1) and (P2) according to the MakeSpan algorithm $\mathbf{GMS}$ given in section 3.2. For any vertex $v \in V$ consider the edges adjacent to v ordered by the length. For any $e \in E_v$, let $S^e(v)$ denote the sum of the length of edges which come before e plus the length of e. For any edge $e = \{u, v\} \in E$, let the bucket value of e be $f^*(e) = \max\{r(e),\ S^e(u),\ S^e(v)\}$. The following two lemmas show the existence of both properties P1 and P2 for the bucketing function f^* and algorithm $\mathbf{GMS}$.

LEMMA 3. *Let* $H = (V', E')$ *be a subgraph of* G*. For any edge* $e = \{u, v\} \in E'$ *there exists an edge* $e^* \in E'$ *such that* $f_{\mathbf{GMS}}(e) \le 3f^*(e^*)$ *and thus satisfying property P1 with* $\beta = 3$*, where* $f_{\mathbf{GMS}}(e)$ *is the finishing time of* e *when we run* $\mathbf{GMS}$ *restricted to* H*. In addition if* e *is released at time zero (i.e.* $r(e) = 0$*) then* $f_{\mathbf{GMS}}(e) \le 2f^*(e^*)$ *for some edge* $e^* \in E'$.

Proof. By Theorem 6 we have $f(e) \le r(e) + \sum_{e'\in E'_u} \ell(e') + \sum_{e'\in E'_v} \ell(e') - \ell(e)$. Assume that p and q are the files with the longest length adjacent to u and v in H, respectively. Thus $S^p(u)$ ($S^q(v)$) is at least the sum of the lengths of all edges in E'_u (E'_v). By the definition of the bucketing function f^* we have

- $f^*(e) \geq r(e)$.
- $f^*(p) \geq S^p(u) \geq \sum_{e' \in E'_u} \ell(e')$,
- $f^*(q) \geq S^q(v) \geq \sum_{e' \in E'_v} \ell(e')$,

Therefore

$$\begin{aligned} f_{\mathbf{GMS}}(e) &\leq r(e) + \sum_{e' \in E'_u} \ell(e') + \sum_{e' \in E'_v} \ell(e') - \ell(e) \\ &\leq f^*(e) + f^*(p) + f^*(q) - \ell(e) \\ &\leq 3f^*(e^*) - \ell(e) \end{aligned}$$

where e^* is the edge with the maximum bucket value among all edges in E'. We note that if $r(e) = 0$ then the above inequality would be $f_{\mathbf{GMS}}(e) \leq 2f^*(e^*) - \ell(e)$. □

LEMMA 4. *The sum of the bucket values of all edges in E is at most 3OPT_{sum} and thus satisfying property P2 with $\gamma = 3$. Furthermore, if all the release times are zero, or if all lengths are uniform, P2 holds with $\gamma = 2$.*

Proof. Let *opt* be the optimum schedule. Consider the finishing times of edges adjacent to an arbitrary vertex $v \in V$ in *opt* and let x_{vi} denote the ith smallest finishing time among them. Note that $\sum_{v \in V} \sum_{i=1}^{deg_v} x_{vi}$ is exactly equal to $2OPT$.

Vertex $v \in V$ should have finished transferring at least k files at time x_{vk} for any $1 \leq k \leq deg_v$. Thus

(i) x_{vk} cannot be less than the sum of lengths of the smallest k edges in E_v. We denote this sum by $sm^k(v)$;

(ii) x_{vk} cannot be less than the kth smallest release time of edges in E_v.

Now we show that $\sum_{e \in E} f^*(e)$ is within 3 factor of OPT. We have:

$$\begin{aligned} \sum_{e \in E} f^*(e) &= \sum_{e=(u,v)} \max\{r(e), S^e(u), S^e(v)\} \\ &\leq \sum_{e=(u,v)} r(e) + \sum_{e=(u,v)} (S^e(u) + S^e(v)) \end{aligned}$$

The first term, $\sum_{e=(u,v)} r(e)$, cannot be more than OPT. The second term is the sum of edges shorter than e adjacent to u or v, when summed over all edges. We can rewrite the second term by summing these values over the edges adjacent to each vertex.

$$\begin{aligned} \sum_{e \in E} f^*(e) &\leq OPT + \sum_{v \in V} \sum_{e \in E_v} S^e(v) \\ &= OPT + \sum_{v \in V} \sum_{k=1}^{deg_v} sm^k(v) \\ &\leq OPT + \sum_{v \in V} \sum_{k=1}^{deg_v} x_{vk} \\ &= 3OPT \end{aligned}$$

We note that if all the release times are zero, then $\sum_{e \in E} f^*(e) \leq 2OPT$. We can prove the same ratio under uniform length assumption.

For a vertex $v \in V$, let $e_{v1}, \ldots, e_{v\, deg_v}$ denote the edges adjacent to v sorted by the release time, breaking ties arbitrarily. We note that since all the lengths are uniform for all $k \leq deg_v$, $S^{e_{vk}}(v) = sm^k(v)$ and thus by (i) and (ii) we have $x_{vk} \geq \max\{r(e_{vk}), S^{e_{vk}}(v)\}$. This shows that under uniform length assumption $\sum_{e \in E} f^*(e)$ cannot be more than $2OPT$.

$$\begin{aligned} \sum_{e \in E} f^*(e) &= \sum_{e=(u,v)} \max\{r(e), S^e(u), S^e(v)\} \\ &\leq \sum_{e=(u,v)} (\max\{r(e), S^e(u)\} + \max\{r(e), S^e(v)\}) \\ &= \sum_{v \in V} \sum_{e \in E_v} \max\{r(e), S^e(v)\} \\ &= \sum_{v \in V} \sum_{k=1}^{deg_v} \max\{r(e_{vk}), S^{e_{vk}}(v)\} \\ &\leq \sum_{v \in V} \sum_{k=1}^{deg_v} x_{vk} \\ &= 2OPT \end{aligned}$$

□

Finally using the meta-algorithm provided in Section 3.1, we get a constant competitive algorithm for the general transfer problem.

THEOREM 7. *For an instance of transfer problem (G, ℓ, r), $\mathbf{ALG}(\mathbf{GMS}, f^*)$ is a $9e$-approximation algorithm for the MinSum problem.*

Proof. By Lemma 3 the bucketing function f^* has (P1) property with $\beta = 3$ and by Lemma 4 has (P2) property with $\gamma = 3$. The claim follows directly from Theorem 5. □

We note that similar to Corollary 2, by Lemma 3 for a zero-release transfer problem or a uniform transfer problem we get property (P1) with $\beta = 2$ which improves the approximation ratio of the algorithm.

COROLLARY 3. *For an instance of zero-release transfer problem (G, ℓ), $\mathbf{ALG}(\mathbf{GMS}, f^*)$ is a $4e$-approximation algorithm for the MinSum problem.*

COROLLARY 4. *For an instance of uniform transfer problem (G, r), $\mathbf{ALG}(\mathbf{GMS}, f^*)$ is a $6e$-approximation algorithm for the MinSum problem.*

4. UNIFORM ZERO-RELEASE TRANSFER MODEL

In this section we consider the uniform zero-release transfer model. First we show that by running all the file transfers simultaneously, we could obtain an exact solution for the MakeSpan version. We call this scheduling algorithm in which $\forall e \in E, s(e) = 0$ by *Simultaneous Start* or simply SS.

THEOREM 8. *For any graph $G = (V, E)$ with uniform zero-release file transfers, $\text{MAX}(SS) = \text{OPT}_{ms}$.*

Proof. Let $\Delta(G)$ be the maximum degree of vertices of G and let u be one of the vertices with degree $\Delta(G)$. The vertex u needs to transfer $\Delta(G)$ units of file and transferring these files takes at least $\Delta(G)$ units of time. So we have $\text{OPT} \geq \Delta(G)$. However, if we start all the jobs simultaneously, in each unit of time we run at least $\frac{1}{\Delta(G)}$ of each transfer. Therefore $\text{MAX}(SS) = \Delta(G) = \text{OPT}$. □

We note that if the files can have arbitrary lengths, SS may not have the optimum cost. We give an example for zero-release non-uniform file transfer model where the cost of SS can be almost two

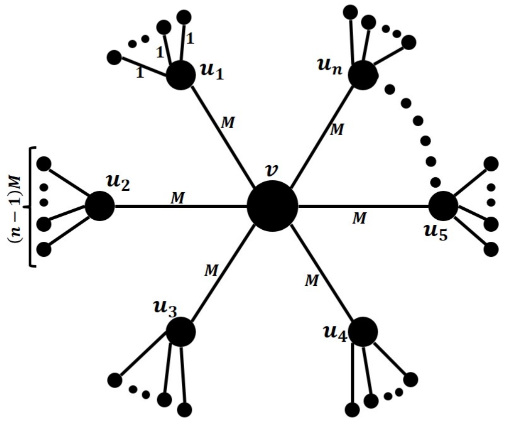


Figure 2: An example where SS is not optimum

times the OPT. Consider the tree shown in Figure 2. Vertex v is adjacent to $n \geq 2$ vertices $u_1, \ldots, u_n$ through the edges $e_1, \ldots, e_n$. The length of all edges adjacent to v is an arbitrary integer M. For every $i \in [n]$ the vertex u_i is adjacent to $(n-1)M$ leaves through the edges of length 1. We can show that $\text{OPT}_{ms} = nM$ but the MakeSpan cost of SS is $\text{MAX}(SS) = 2nM - (M + n - 1)$.

The optimum schedule has n stages. In stage i vertex u_i starts transferring e_i and for every $j \neq i$ the vertex u_j starts transferring with M of its adjacent leaves. Therefore each stage takes exactly M units of time and OPT_{ms} would be equal to nM (the sum of edges adjacent to v is nM and thus OPT cannot be smaller).

The schedule SS starts all edges simultaneously at time zero. Since all edges are adjacent to a vertex with degree $(n-1)M+1$, the edges adjacent to the leaves finish at time $(n-1)M+1$. After that the remaining $M-1$ units of data on edges adjacent to v will be transferred with speed of $1/n$ and SS finishes all transfers at time $(n-1)M + 1 + (M-1)n$. Therefore by choosing $n = M$ the ratio between the optimum cost and the cost of SS would be

$$\frac{\text{MAX}(SS)}{\text{OPT}_{ms}} \geq \frac{2n^2 - 2n}{n^2} = 2\frac{n-1}{n}.$$

Now we show there is always a non-concurrent schedule with a cost less than twice the optimum solution in the MinSum problem. First we need to give a lower bound on the optimum cost.

LEMMA 5. *For any graph $G = (V, E)$ with uniform zero-release file transfers,* $\text{OPT}_{sum} \geq \frac{1}{4}\sum_{v \in V}(deg_v^2 + deg_v)$.

Proof. Let *opt* be the optimum schedule. Consider the finishing times of edges adjacent to an arbitrary vertex $v \in V$ in *opt* and let x_{vi} denote the ith smallest finishing time among them. Since $x_{vi} \geq i$ for all $1 \leq i \leq deg_v$, we have $\sum_{e \in E_v} f_{opt}(e) = \sum_{i=1}^{deg_v} x_{vi} \geq \frac{deg_v(deg_v+1)}{2}$. Therefore

$$\begin{aligned} 2\text{OPT} &= 2\sum_{e \in E} f_{opt}(e) \\ &= \sum_{v \in V}\sum_{e \in E_v} f_{opt}(e) \geq \sum_{v \in V}\frac{deg_v^2 + deg_v}{2}. \end{aligned}$$

□

THEOREM 9. *For any graph $G = (V, E)$ with uniform zero-release file transfers, there is a non-concurrent schedule $\mathcal{S}$ with* $\text{SUM}(\mathcal{S}) \leq 2\text{OPT}_{sum}$ *with integral starting times.*

Proof. Let *opt* be the optimum schedule for G. Assume that the set of edges $e_1, e_2, \ldots, e_{|E|}$ are sorted according to their finishing time in *opt*, i.e., $f_{opt}(e_1) \leq f_{opt}(e_2) \leq \cdots \leq f_{opt}(e_{|E|})$. Now we make a non-concurrent schedule $\mathcal{S}$ based on this sequence with the total cost at most 2OPT.

For any vertex $v \in V$ and $1 \leq i \leq |E|$, let E_u^i be the subset of edges from $e_1, \ldots, e_i$ which are adjacent to u. We start transferring edge $e_i = \{u, v\}$, whenever both its endpoints have finished transferring E_u^{i-1} and E_v^{i-1}. Formally, the starting time for edge e_i is

$$s_{\mathcal{S}}(e_i) = \min\{t \mid t \in \mathbb{Z}^{\geq 0}, \forall e \in E_u^{i-1} \cup E_v^{i-1}\ [t \neq s_{\mathcal{S}}(e)]\}.$$

By definition of $s_{\mathcal{S}}(e_i)$, it is clear that *(i)* the resulting schedule is a non-concurrent schedule, therefore $f_{\mathcal{S}}(e_i) = s_{\mathcal{S}}(e_i) + 1$; and *(ii)* $s_{\mathcal{S}}(e_i) \leq |E_u^{i-1}| + |E_v^{i-1}|$ which means that $f_{\mathcal{S}}(e_i) \leq |E_u^i| + |E_v^i| - 1$. We know in *opt*, the finishing times of all the edges in E_u^i and E_v^i are less than or equal to $f_{opt}(e_i)$. Since the speed of transfer is at most 1 for any processor, vertex u cannot finish all the edges of E_u^i in less than $|E_u^i|$; hence $f_{opt}(e_i) \geq \max\{|E_u^i|, |E_v^i|\}$. Therefore for every edge e_i we have $f_{\mathcal{S}}(e_i) \leq 2f_{opt}(e_i)$ and so $\text{SUM}(\mathcal{S}) \leq 2\text{OPT}$. □

COROLLARY 5. *The uniform file transfer problem without release times on planar graphs admits a $2 + \epsilon$ approximation ratio.*

This follows from the non-concurrent PTAS of Marx [23] for coloring the edges of a planar graph.

In uniform transfer model, if all the starting times are integers in a non-concurrent schedule, the schedule is indeed a partition of the edges E, into k matchings $M_1, \ldots, M_k$ for some k, where M_i is the set of all edges starting on time $i-1$ and therefore finishing at time i. The cost of this schedule would be $\sum_{1 \leq i \leq k} i|M_i|$. Halldórsson et. al. [2] give an approximation algorithm of ratio 1.8289 for finding the minimum sum cost edge coloring. By Theorem 9, the same algorithm gives us a 3.658-approximation algorithm for the zero-release uniform transfer model.

COROLLARY 6. *There is a polynomial time algorithm which gives a non-concurrent schedule with the cost less than 3.658-factor of* OPT_{sum} *for the zero-release uniform transfer model.*

5. FILE TRANSFER ON BIPARTITE GRAPHS

For certain classes of graphs we may get a smaller constant approximation. Consider a graph $G = (V, E)$ and a non-concurrent schedule of E into k matchings $M_1, \ldots, M_k$. If for every vertex $v \in V$ all edges adjacent to v are in the first deg_v matchings, then the sum of edges adjacent to v would be $\sum_{i=1}^{deg_v} i = deg_v(deg_v+1)/2$. Therefore the sum of finishing times over all edges would be $\frac{1}{2}\sum_{v \in V} deg_v(deg_v+1)/2$ which by Lemma 5 is equal to the optimum cost. For example, in k-regular bipartite graphs we can always find such a schedule by simply partitioning the edges into k perfect matchings, thus:

COROLLARY 7. *For any regular bipartite graph G we can find a schedule with the cost equal to* OPT_{sum} *in polynomial time.*

The nature of many transfer problems are transferring files between hosts and clients thus bipartite graphs are of separate interest.

We present a 1.414-approximation algorithm for finding a schedule with minimum sum cost in bipartite graphs. We rely on the algorithm [5] in non-concurrent settings but we need to change it somewhat. We first argue a simple ratio 2 approximation MinSum file transfer of unit jobs on bipartite graphs (without release time). Then we show a $\sqrt{2}$ approximation based on [5] for the sum version.

Let $G = (V, E)$ be a graph. We say vertex v is *full* in G when $deg_v = \Delta(G)$[3]. A graph G is in *class 1* iff $\chi'(G) = \Delta(G)$ where $\chi'(G)$ is the edge-chromatic number of G. Theorem 17.2 of [1] shows any bipartite graph G is in *class 1* and can be partitioned into $\chi'(G)$ matchings in polynomial time. Assume an instance of the uniform zero-release file transfer problem with a bipartite graph $G = (V, E)$. Since $\chi'(G) = \Delta(G)$ we can partition the edge set E, into $\Delta(G)$ matchings. Consider M as one of these matchings. If we remove the edges of M from the graph, the resulting graph would have an edge chromatic number of $\Delta(G) - 1$ and thus the maximum degree of $\Delta(G) - 1$ (since the resulting graph is also in class 1). Therefore every full vertex v in G must have one edge in M. Let $E_{\Delta(G)}$ be the subset of the edges of M which are adjacent to at least one full vertex in G.

Let $G^{\Delta(G)-1} = (V, E \backslash E_{\Delta(G)})$. Since every full vertex in G has an edge in $E_{\Delta(G)}$ we have $\Delta(G^{\Delta(G)-1}) = \Delta(G) - 1$. With the same argument we can find a matching $E_{\Delta(G)-1}$ in $G^{\Delta(G)-1}$, such that every full vertex in $G^{\Delta(G)-1}$ has one adjacent edge in that matching, and each edge in that matching is adjacent to at least one full vertex in $G^{\Delta(G)-1}$.

Now the graph $G^{\Delta(G)-2} = \big(V, E \backslash (E_{\Delta(G)} \cup E_{\Delta(G)-1})\big)$ has the maximum degree of $\Delta(G) - 2$. By repeating this procedure we partition E into $\Delta(G)$ matchings $E_{\Delta(G)}, E_{\Delta(G)-1}, \ldots, E_1$.

Algorithm 3

Input: An instance of zero-release uniform file transfer problem $G = (V, E)$ where G is bipartite.
Output: A schedule $\mathcal{S}$ with corresponding starting times for each edge in E.

```
Define G^{Δ(G)} = G.
for i = Δ(G) to 1 do
    Partition the edges of G^i into χ'(G^i) = Δ(G^i) matchings
    and let M be one of the matchings.
    Let V^i_full ⊆ V be the set full vertices in G^i.
    Let E_i ⊆ M be the subset of edges in M which are adjacent
    to at least one of the vertices in V^i_full.
    For any edge e in E_i, set s_S(e) = i − 1.
    Define G^{i−1} = (V, E \ ⋃_{j=i}^{Δ(G)} E_j).
```

It can easily be shown that Algorithm 3 is a 2-approximation algorithm.

THEOREM 10. *Algorithm 3 is a 2-approximation algorithm for the MinSum cost in uniform zero-release file transfer problem in bipartite graphs.*

Proof. Consider the bipartite graph $G = (V, E)$. Running Algorithm 3 gives us a schedule $\mathcal{S}$. The schedule $\mathcal{S}$ partitions the edges into $\Delta(G)$ matchings $E_1, \ldots, E_{\Delta(G)}$ such that for any i, $1 \le i \le \Delta(G)$, any full vertex in G^i has an edge in E_i and any edge in E_i is adjacent to at least one full vertex in G^i (where $G^i = (V, E \backslash \bigcup_{j=i+1}^{\Delta(G)} E_j)$). Let $V^i_{full} \subseteq V$ be the set of full vertices in G^i. Since each edge in E_i is adjacent to at least one vertex in V^i_{full} we have $|E_i| \le |V^i_{full}|$.

Let $n(i)$ denote the number of vertices in G with degree at least i, i.e., $n(i) = |\{v \in V | deg_v \ge i\}|$. Recall that a vertex is full in G^i iff the degree of v in G^i is equal to $\Delta(G^i)$. The maximum degree $\Delta(G^i)$ is equal to i since G^i is the union of i matchings and thus the degree of any full vertex in G^i would be at least i in G. Therefore $|V^i_{full}| \le n(i)$. Considering the sum cost of the schedule we have

$$\begin{aligned}
\textsc{sum}(\mathcal{S}) &= \sum_{1 \le i \le \Delta(G)} i\,|E_i| \\
&\le \sum_{1 \le i \le \Delta(G)} i\,|V^i_{full}| \le \sum_{1 \le i \le \Delta(G)} i\,n(i) \\
&= \sum_{1 \le i \le \Delta(G)} i \sum_{\{v | deg_v \ge i\}} 1 = \sum_{v \in V} \sum_{1 \le i \le deg(v)} i \\
&= \sum_{v \in V} \frac{deg(v)(deg(v)+1)}{2} \le 2\textsc{opt}_{sum}.
\end{aligned}$$

The last line is the result of Lemma 5. □

In [5] it is shown that the algorithm is $\sqrt{2}$-approximate and there analysis is almost tight. One can change their proof to obtain the same approximation ratio. The proof is presented in the next subsection.

COROLLARY 8. *There is a polynomial time algorithm which gives a non-concurrent schedule with the cost less than $\sqrt{2}$-factor of $\textsc{opt}_{sum}$ for the zero-release uniform transfer problem in bipartite graphs.*

5.1 Approximation Ratio of Algorithm 3

Gandhi and Mestre [5] use a class of matchings which are *strongly minimal*. One can prove that the same property given is sufficient for a scheduling to be $\sqrt{2}$-approximate in the uniform zero-release file transfer model. Not all graphs admit strongly minimal matchings but in some classes of graphs such as bipartite graphs we can give polynomial time algorithms to find such matchings. For the sake of completeness we present a slightly modified proof.

Let $G = (V, E)$ and let $\mathcal{S}$ be a non-concurrent schedule of E. Recall that $\mathcal{S}$ can be shown as the partition of E into matchings $M_1, \ldots, M_k$. For i, $1 \le i \le k$, let $G^i = \left(V, \bigcup_{j=1}^{i} M_i\right)$ (thus $G^k = G$). Let deg^i_v for a vertex $v \in V$ and $1 \le i \le k$ denote the degree of vertex v in G^i. By borrowing notations from [5], we say vertex v is *full* in G^i when $deg^i_v = \Delta(G^i)$.

We call $\mathcal{S}$ *strongly minimal* if for every i, $1 \le i \le k$:

- For every full vertex v in G^i, there is one edge adjacent to v in M_i.
- At least one of the endpoints of every edge of E_i is full in G^i.

Another way to look at this property is that G^i is a maximal i-matching w.r.t G for every $i \le k$. We note that by definition of strongly minimal schedule the number of matchings k is indeed equal to $\Delta(G)$. Furthermore, since every full vertex in E^i has exactly one edge in M_i we have $\Delta(G^{i-1}) = \Delta(G^i) - 1$ and in general for every i, $1 \le i \le \Delta(G)$, $\Delta(G^i) = i$. Thus if v is full in G^i for some i, then it is full in G^j for every $j \le i$.

THEOREM 11. *For $G = (V, E)$ in the uniform zero-release file transfer model any strongly minimal schedule $\mathcal{S}$ is $\sqrt{2}$-approximate.*

[3]For a graph H, $\Delta(H)$ is the maximum degree in H

Proof. The idea is to make a full vertex in G^i responsible for paying the finishing time of its adjacent edge in E^i. Formally, for an edge $e = \{u, v\}$ assume that that e is in the matching M_i. By the definition of $\mathcal{S}$ at lease one of u or v is full in G^i. If both endpoints are full then each of u and v are *half-responsible* for e. If only one of the endpoints, say u, is full then u is *fully-responsible* for e.

Now consider an arbitrary vertex $v \in V$. Assume that v is fully-responsible and half-responsible for the set of edges R_v^1 and R_v^2 respectively. Define $C_{opt}(v) = \sum_{e \in R_v^1} f_{opt}(e) + \sum_{e \in R_v^2} \frac{f_{opt}(e)}{2}$ where *opt* is the optimum schedule for G. Similarly define $C_{\mathcal{S}}(v) = \sum_{e \in R_v^1} f_{\mathcal{S}}(e) + \sum_{e \in R_v^2} \frac{f_{\mathcal{S}}(e)}{2}$. In other words v always pays the finishing times of edges in R_v^1 and pays the half of the finishing times of edges in R_v^2. Thus $\text{SUM}(opt) = \sum_v C_{opt}(v)$ and $\text{SUM}(\mathcal{S}) = \sum_v C_{\mathcal{S}}(v)$.

To show that $\mathcal{S}$ is α-approximate it is sufficient to show that $\frac{C_{\mathcal{S}}(v)}{C_{opt}(v)} \leq \alpha$. Fix a vertex v and let $n_1 = |R_v^1|$ and $n_2 = |R_v^2|$. Vertex v is full in $G^1, \ldots, G^{n_1+n_2}$ since each vertex gets at least a half responsibility when it is full in some G^i. Thus for an edge $e \in R_v^1 \cup R_v^2$ the finishing time of e in $\mathcal{S}$ is not greater than $n_1 + n_2$. More precisely, the set of finishing times of edges in $R_v^1 \cup R_v^2$ is exactly $\{1, \ldots, n_1 + n_2\}$. Therefore

$$\begin{aligned} C_{\mathcal{S}}(v) &= \sum_{e \in R_v^1} f_{\mathcal{S}}(e) + \sum_{e \in R_v^2} \frac{f_{\mathcal{S}}(e)}{2} \\ &= \sum_{e \in R_v^1 \cup R_v^2} f_{\mathcal{S}}(e) - \sum_{e \in R_v^2} \frac{f_{\mathcal{S}}(e)}{2} \\ &= \sum_{i=1}^{n_1+n_2} i - \sum_{e \in R_v^2} \frac{f_{\mathcal{S}}(e)}{2} \\ &\leq \sum_{i=1}^{n_1+n_2} i - \sum_{i=1}^{n_2} \frac{i}{2} \\ &= \frac{(n_1+n_2)(n_1+n_2+1)}{2} - \frac{(n_2)(n_2+1)}{4} \\ &= \frac{n_1^2+n_1}{2} + \frac{n_2^2+n_2}{4} + n_1 n_2. \end{aligned}$$

Since the edges in R_v^1 and R_v^2 are all adjacent *opt* cannot finish transferring all of them sooner than $n_1 + n_2$, thus

$$\begin{aligned} C_{opt}(v) &= \sum_{e \in R_v^1} f_{opt}(e) + \sum_{e \in R_v^2} \frac{f_{opt}(e)}{2} \\ &= \sum_{e \in R_v^1 \cup R_v^2} \frac{f_{opt}(e)}{2} + \sum_{e \in R_v^1} \frac{f_{opt}(e)}{2} \\ &\geq \sum_{i=1}^{n_1+n_2} \frac{i}{2} + \sum_{e \in R_v^1} \frac{f_{opt}(e)}{2} \\ &\geq \sum_{i=1}^{n_1+n_2} \frac{i}{2} + \sum_{i=1}^{n_1} \frac{i}{2} \\ &= \frac{(n_1+n_2)(n_1+n_2+1)}{4} + \frac{(n_1)(n_1+1)}{4} \\ &= \frac{n_1^2+n_1}{2} + \frac{n_2^2+n_2}{2} + \frac{n_1 n_2}{2}. \end{aligned}$$

We need to determine the smallest α such that

$$\alpha \geq \frac{(n_1^2+n_1) + (n_2^2+n_2)/2 + 2(n_1 n_2)}{(n_1^2+n_1) + (n_2^2+n_2)/2 + (n_1 n_2)} \geq \frac{2n_1^2 + n_2^2 + 4n_1 n_2}{2n_1^2 + n_2^2 + 2n_1 n_2}.$$

By changing the variable to $x = \frac{n_1}{n_1+n_2}$ we get

$$\alpha \geq \frac{1 + 2x - x^2}{1 + x^2}.$$

Finally the right hand side is maximized for $x = \sqrt{2} - 1$, which gives us $\alpha \geq \sqrt{2}$. □

Running Algorithm 3 on a bipartite graph $G = (V, E)$ gives us a schedule $\mathcal{S}$. The schedule $\mathcal{S}$ partitions the edges into $\Delta(G)$ matchings $E_1, \ldots, E_{\Delta(G)}$ such that for any i, $1 \leq i \leq \Delta(G)$, any full vertex in G^i has an edge in E_i and any edge in E_i is adjacent to at least one full vertex in G^i (where $G^i = (V, E \setminus \bigcup_{j=i+1}^{\Delta(G)} E_j)$). This shows that the schedule given by Algorithm 3 is strongly minimal and thus proves Corollary 8.

6. CONCLUSION AND OPEN PROBLEMS

This paper studies a local protocol for file transfer problems which to the best of our knowledge, has not been studied before in theoretical computer science. Among the problems we consider, we highlight one open problem of primary concern: Is there a gap between the optimum concurrent schedule and optimum non-concurrent schedule when minimizing the average finishing time in the case of the zero-release file transfer model? In this paper most of our algorithms give non-concurrent solutions paying a constant factor compared to the optimum concurrent schedule. It would be instructive to see whether we can design concurrent algorithms with better approximation factors. This shows the importance of finding the gap between optimum concurrent and optimum non-concurrent schedules.

It would also be interesting to consider the online version of the problem where the release times are revealed to the algorithm in an online fashion. Could we get constant approximations in non-preemptive model or do we need to add preemptive assumptions to the problem?

7. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

8. REFERENCES

[1] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics, 244. Springer, New York, 2008.

[2] M.M. Halldórsson, G. Kortsarz and M. Sviridenko. *Min Sum Edge Coloring in Multigraphs Via Configuration LP*. In Proc. 13th Conf. Integer Prog. Combin. Optimiz. (IPCO), 2008.

[3] R. Gandhi, M. M. Halldórsson, G. Kortsarz and H. Shachnai. *Improved Bounds for Scheduling Conflicting Jobs with Minsum Criteria*. ACM Transactions on Algorithms. Vol. 4, No. 1, 2008.

[4] M.M. Halldórsson and G. Kortsarz. *Tools for multicoloring with applications to planar graphs and partial k-trees*. Journal of Algorithms 42, 2, 334-366, 2002.

[5] R. Gandhi and J. Mestre. *Combinatorial Algorithms for Data Migration to Minimize Average Completion Time*. Algorithmica 54, 1,pp 54-71, 2009.

[6] Y. Kim. *Data Migration to Minimize the Average Completion Time*. Journal of Algorithms,55:42-57, 2005.

[7] M.K. Goldberg, *Edge-coloring of multigraphs: recoloring technique*. J. Graph Theory, 8:121-137, 1984

[8] D.S. Hochbaum, T. Nishizeki, and D.B. Shmoys. *A better than "Best Possible" algorithm to edge color multigraphs*. Journal of Algorithm 7:79-104, 1986.

[9] E. G. Coffman, M. R. Garey, D. S. Johnson, and A. S. Lapaugh. *Scheduling file transfers*. SIAM Journal on Computing, 14(3):744-780, 1985.

[10] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. *On chromatic sums and distributed resource allocation*. Information and Computation, Vol. 140, pp. 183-202, 1998.

[11] R. Gandhi, M. M. Halldórsson, G. Kortsarz, and H. Shachnai. *Improved Results for Data Migration and Openshop Scheduling*. ACM Transactions on Algorithms, 2(1):116-129, 2006.

[12] M. M. Halldorsson, G. Kortsarz, and H. Shachnai. *Sum Coloring Interval Graphs and k-Claw Free Graphs with Applications for Scheduling Dependent Jobs*. Algorithmica, 37:187-209, 2003.

[13] P. Sanders and D. Steurer. *An Asymptotic Approximation Scheme for Multigraph Edge Coloring*. Proc. of the 16th ACM-SIAM Symposium on Discrete Algorithms, 2005.

[14] R. Graham. *Bounds for certain multiprocessing anomalies*. Bell System Technical Journal, 45:15631581, 1966.

[15] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. *Improved Scheduling Problems For Minsum Criteria*. Proc. of the 23rd International Colloquium on Automata, Languages, and Programming, LNCS 1099, 646-657, 1996.

[16] H. Hoogeveen, P. Schuurman, and G. Woeginger. *Non-approximability Results For Scheduling Problems with Minsum Criteria*. Proc. of the 6th International Conference on Integer Programming and Combinatorial Optimization, LNCS 1412, 353-366, 1998.

[17] M. Queyranne and M. Sviridenko. *A* $(2 + \epsilon)$*-Approximation Algorithm for Generalized Preemptive Open Shop Problem with Minsum Objective*. Journal of Algorithms, 45:202-212, 2002.

[18] M. Queyranne and M. Sviridenko. *Approximation Algorithms for Shop Scheduling Problems with Minsum Objective*. Journal of Scheduling, 5:287-305, 2002.

[19] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. *An Experimental Study of Data Migration Algorithms*. Workshop on Algorithm Engineering, pages 145-158, 2001.

[20] J. Hall, J. Hartline, A. Karlin, J. Saia, and J. Wilkes. *On Algorithms for Effcient Data Migration*. Proc. of the 12th ACM-SIAM Symposium on Discrete Algorithms, pages 620-629, 2001.

[21] S. Khuller, Y. Kim, and Y. C. Wan. *Algorithms for Data Migration with Cloning*. In Proc. of the 22nd ACM Symposium on Principles of Database Systems, pages 27-36, 2003.

[22] Dániel Marx, *Minimum sum multicoloring on the edges of trees*. Theor. Comput. Sci., volume 361, number 2-3, pages 133-149, 2006.

[23] Dániel Marx, *Minimum sum multicoloring on the edges of planar graphs*. WAOA, pages 9-22, 2004

[24] , M. Garey and D. Johnson. *Computer and intractability. A guide to the theory of NP-completeness* Freeman, 1979.

[25] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. In Proc. of the 6th Symposium on Operating System Design and Implementation, pages 137-150, 2004.