Scheduling in Bandwidth Constrained Tree Networks [Extended Abstract]

Sungjin Im
University of California, Merced
Merced, CA 95343
sim3@ucmerced.edu

Benjamin Moseley
Washington University in St. Louis.
St. Louis, MO 63130
bmoseley@wustl.edu

ABSTRACT

In this paper we introduce a new network scheduling model. Here jobs need to be sent via routers on a tree to machines to be scheduled, and the communication is constrained by network bandwidth. The scheduler coordinates network communication and job machine scheduling. This type of scheduler is highly desirable in practice; yet few works have considered combing networking with job processing. We consider the popular objective of total flow time in the online setting. We give a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^7})$-competitive algorithm when all routers are identical and all machines are identical for any fixed $\epsilon > 0$. Then we go on to show a $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^7})$-competitive algorithm when the routers are identical and the machines are *unrelated*. To show these results we introduce an interesting combination of potential function and dual fitting techniques as well as a reduction of general tree scheduling to a special case of trees.

Categories and Subject Descriptors

F.2.2 [**Nonnumerical Algorithms and Problem**]: Sequencing and scheduling

General Terms

Algorithms, Theory

Keywords

Online scheduling, Flow time, Bandwidth, Tree network.

1. INTRODUCTION

Scheduling jobs online in a distributed computing environment is fundamental to a variety of applications in practice. Due to the essential nature of multiple machines scheduling, there has been a continual effort to design efficient scheduling algorithms for such systems. See [33] for pointers to previous work. In the most basic multiple machine environment there are n jobs which arrive over time online that are to be scheduled on a set of identical machines. In the *online* setting, the scheduler becomes aware of a job only when it arrives. Most systems require online schedulers because they typically do not know of a job until the client submits the job to the system. In the *identical* machines setting each job J_j requires processing time p_j and the processing time of a job is the same on any machine. The identical machine model is the most basic multiple machine model.

The identical machine setting, although widely studied and an important model, does not capture a variety of systems seen today in practice. Indeed, machines can have different processor speeds, amounts of memory, I/O devices, and even consume different amounts of energy. Due to this, a variety of generalized models have been addressed in scheduling theory. One such model is the related machine model. In the *related* machines setting each machine i runs at some fixed speed s_i and the processing requirement of a job J_j on a machine is p_j/s_i. This captures the case where machines have different processor speeds. However, some job's processing time may not only depend on processor speeds. Indeed, a job has multiple resources it requires (e.g. I/O devices, memory). Due to the multiple dimensions of resources a job requires, the *unrelated* machine model has been considered. In the unrelated machine model, a job J_j's processing requirement on machine i is $p_{j,i}$. The processing time of a job can be arbitrarily different between machines.

There has been an extensive study of scheduling jobs online in these machine environments. For example, see [3, 6, 12–14, 18, 19, 30]. It is fair to say that it has been challenging to develop algorithms with strong guarantees for the more general models. This line of work has had two goals. One is to understand the online complexity of each of these basic scheduling models and to find good algorithms which can be used in practice.

Unfortunately, these multiple machine models have made the unrealistic assumption that a job can be immediately sent to any machine and start being processed instantaneously on the machine. This is generally not the case in practice. Indeed, jobs typically require access to data before they can be started on a machine. The job requires its data to be moved from its current location to the machine that schedules the job. This can be a main performance bottleneck in practice because off-site memory accesses take several orders of magnitude more time than local memory accesses and computation. For a scheduler to be useful in practice, it would need to incorporate this time into its scheduling decisions. Indeed, many distributed systems today are used for large data analysis and, when data sets are

SPAA'15, June 13–15, 2015, Portland, OR, USA.

http://dx.doi.org/10.1145/2755573.2755576.

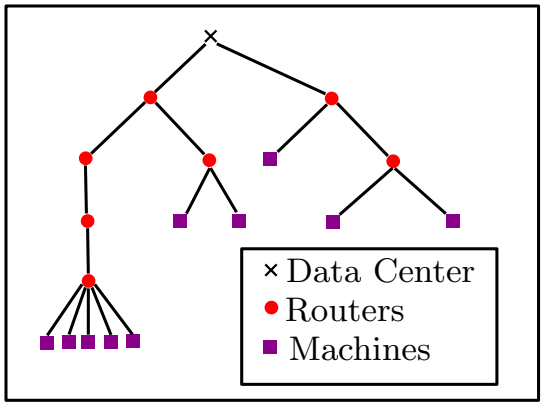

Figure 1: Tree network

large, the main time constraint for job processing is moving the massive data between the machines. For instance this is the case in MapReduce and Hadoop environments [27, 31]. Since this is a main bottleneck in systems, algorithms developed for multiple machine scheduling may not be used since they do not take this constraint into consideration.

This type of scheduling constraint has been referred to as a communication or networking constraint. Addressing these scheduling issues is not new. Besides a large amount of work being done in practical scheduling literature, there has been some theoretical work done offline with the goal of minimizing makespan [8, 9, 16, 17, 20, 21]. Additionally, over two decades ago Phillps, Stein and Wein [32] introduced an interesting model of network scheduling and argued its importance in practice. In their model, before a job can be started on a machine, the job and its data need to be moved to the machine. Here there is a graph that induces the time a job requires to be moved to a machine. When moving a job's data through the network, jobs do not conflict with each other. That is, jobs can freely share links in the network. In their model, this essentially results in jobs having different 'arrival' times depending on the machine they are scheduled on.

Since the work of [32], even in the offline setting, there has been essentially no work which takes networking constraints into consideration for flow objectives such as total flow time. Further there has been essentially no previous work in the online setting. No model has been introduced where there is *congestion* constraints in the network. That is, if a job's data is being moved through a network then a scheduler will need to prioritize which job's data is moved across a particular link or router at a moment in time. Perhaps the reason previous work has not considered networking constraints is because scheduling on multiple machines alone is quite challenging in itself and it is not clear that positive results can be shown for such settings or even what an algorithm might look like.

Tree Network Model: In this paper we consider a network scheduling problem with congestion constraints on tree networks. Trees are among the most popular network topologies [1, 15]. This is because trees scale well to large networks [2, 23]. In this problem we are given a rooted tree T. It is assumed that the root node of the tree is the job distribution center. The leaves of the tree are machines which perform the job processing. A job must be assigned to a leaf for processing, which is decided by the scheduler. Each of the interior nodes of the tree network is a router. A job's data need to be routed from the root to the leaf machine it is to be scheduled on. A job J_j's data has some size p_j. If all of the nodes of the tree are identical, then J_j requires p_j time steps on a link to send its data to another router. Each link can only move one job at each time and cannot send a job until all of a job's data has been received from the previous router. Generally, the different nodes of the tree could run at different speeds to capture different router speeds. Finally, once a job's data has been moved to a leaf node, the job can be processed at this node. The machines at the leaves could be modeled using identical, related or unrelated machines. See Figure 1. This model captures the widely used tree network topologies seen in practice. Further, it can be seen that this model captures bus topologies where offsite data is routed along a bus to machines the job is to be scheduled on. A natural generalization of our models is to allow jobs to be created at different machines (leaves) in the tree. This extension seems challenging and we leave this to future work.

Results: In this paper we initiate the study of scheduling jobs online in the tree network model. One of the main contributions of this paper is initiating the study of schedulers that coordinate scheduling on a network and on machines online. The objective function we consider is minimizing total (average) flow time. The flow time of a job is the amount of time the job waits in the system until it is satisfied. By minimizing the total flow time, the scheduler focuses on optimizing the average quality of service. This is perhaps the most popular objective considered in scheduling theory. When scheduling jobs in multiple machine environments, it is known that strong lower bounds exist online and offline even in the most basic special cases of the problems we consider [30]. Due to this, we consider the popular resource augmentation model introduced in [26] where our algorithm is given extra speed over the adversary. We say an algorithm is s-speed c-competitive if each router or machine can run up to s times faster than that routers/machines in the adversary's schedule and the algorithm achieves a competitive ratio of c. For all of our results, we consider non-migratory algorithms that compare against a non-migratory adversary. A *non-migratory* algorithm processes a job only on one machine (one leaf machine in our model). Our algorithms will also be immediate-dispatch. An *immediate-dispatch* algorithm decides the leaf that will process a job as soon as the job arrives.

In the online setting we consider two cases. The first case we consider we call the identical endpoint case. Here each job J_j requires p_j time units to be completed on any leaf machine it is assigned to and further the job requires p_j time units to be sent to any router. We consider this model because it essentially captures just the networking aspect of the tree network model. Here we show the following theorem.

Theorem 1 *In the tree network model when all routers and machines are identical there exists a* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^7})$*-competitive algorithm for any* $\epsilon > 0$.

It is known that any algorithm is $\omega(1)$-competitive even when scheduling on identical machines [30], a special case of

the identical endpoint model, thus this is essentially the best positive result one can hope for in this setting using worst case analysis. We then go on to consider a more general setting, which we call the unrelated endpoints setting. Here all the processing requirement on each of the routers for a job is identical. However, the processing times of each job on the leaf machines are *unrelated* (can be completely different). Thus, this is a combination of a network with identical routers and unrelated machines environment where processing times of a job (on the leaves) are unrelated between machines. Here we show the following.

Theorem 2 *In the tree network model when all routers are identical and machines are unrelated there exists a* $(2+\epsilon)$*-speed* $O(\frac{1}{\epsilon^2})$*-competitive algorithm for any* $\epsilon > 0$.

Techniques: An overview of our algorithms and analysis is given in Section 3.1 and here we just remark on some techniques used. In the tree network setting, it is fairly difficult to adapt existing scheduling techniques. This is because we have a layered scheduling problem where scheduling decisions made on one router will later affect decisions on routers further down the tree. Due to this, we will not have the structure that is usually present when scheduling on multiple machines, such as having a simple expression for what a job's flow time will be assuming no more jobs arrive. This is usually very important to potential function or dual fitting analysis. See [24] for a survey on potential functions for scheduling analysis and [3,22] for recent developments on dual fitting for scheduling. Due to this, we need to develop new techniques. We introduce a new potential type analysis to determine an approximation of a job's waiting time at any point in time. Then we consider reducing a general tree to a simpler more structured tree *online*. We then combine these two ideas in conjunction with an online primal-dual analysis of our algorithm on the simpler tree. Then we show that if our algorithm works well on the simpler tree, we can generate *online* an algorithm which works well for any tree.

Related Work: Scheduling on multiple machine was first considered on identical machines. For this setting $O(\min\{\log P,\ \log(n/m)\})$-competitive algorithm is known for minimizing total flow time and there is a matching lower bound online [30]. Here P is the ratio of the maximum to minimum processing time of a job. Assuming resource augmentation a series of works [3, 12–14, 18] has culminated in a $(1+\epsilon)$-speed $O(1/\epsilon)$-competitive algorithm for unrelated machines.

As stated, our model is related to packet routing. Most previous work on packet routing has focused on two models. The first model studies stability of a network where the goal is to keep the number of packets in the system bounded when the system runs for an arbitrary long period of time. See [4, 10,11] for example and pointers to relevant work. The other model is where routers have a fixed buffer size at each of the routers and packets can be dropped when the buffer becomes full. The goal is to maximize the total throughput, i.e. the number of packets sent [7, 28, 34]. These works differ from ours because we requires all job (packets) to be completely processed and we consider the flow time objective over all the packets. One recent work [5] has considered minimizing total flow time when routing in a *line* network. Here for total flow time, no positive results were shown, but it was shown that no algorithm can be $O(1)$-competitive, giving evidence that the problem is algorithmically challenging. For minimizing the maximum flow time of packets on a line they give a $(1+\epsilon)$-speed $O(1)$-competitive algorithm for any fixed $\epsilon > 0$. In the offline setting, packet routing was studied when each packet needs to be routed along its own path, but the goal was to route all packets as early as possible. For example, see the seminal work in [29].

2. PRELIMINARIES

In this section we introduce notation and the problem formally. We will be given a rooted tree T on m nodes and n jobs arrive over time. A job J_i arrives at time r_i. We will assume without loss of generality that all jobs arrive at distinct times. Each of these jobs needs to be processed by some leaf machine/node in T. The algorithm decides the leaf assignment. To be processed on a leaf machine, the job must have its data transferred from the root to the leaf machine before it can start being processed. A job's data must be transferred through links in the network. No more than one job can use a fixed edge/link at any moment in time. For simplicity of explanation, we can view this as a job requiring processing at each node in the tree on the path to the leaf it is assigned to. Thus, we no longer discuss jobs on edges. Further, a job cannot be processed by any node until it is processed by its parent and no job needs to be processed by the root node and no leaf is adjacent to the root. We assume congestion is at the links, thus a job need not be processed by the root node in this view. Note that the problem where congestion happens at the routers can be captured by our setting as well.

One may ask why we consider a model where a job cannot be processed on a router until it is fully processed by the parent router. The main reason this is presented in the paper is that this is a more challenging setting to analyze. The challenge is because extra congestion can happen at internal routers in the network and this effect is effectively negated when the jobs can be divided into unit sized packets when routing. In fact, all the results in this paper can be extended in a fairly straightforward manner to the case where jobs can be sent in small pieces on the routers to obtain similar results in both the identical and unrelated endpoint settings. The details of this are omitted and will appear in a full version of this paper. Also, we consider this case because it captures packet routing, one application of our model. In many packet routing settings packets must be sent completely to a router before it can be forwarded. To see how this fits in our setting, you can imagine packets of data originating at a data collection site that must be transferred to machines to be processed. We note that even in the simplest setting for packet forwarding of unit time packets there are no known results for flow time on tree networks.

We will consider two models for how the job can be processed in the tree. In the identical node setting, a job requires p_j units of processing on every node. In the unrelated endpoint setting, a job requires p_j units of processing on nodes which are not leaf nodes. On a leaf node v, a job requires $p_{j,v}$ units of processing which can be arbitrarily different depending on the leaf.

We will be considering algorithms in the resource augmentation setting, where the algorithm is given extra resources over the adversary. We will give our nodes extra speed over

the adversary. In some case, we may only augment the speed of some nodes in the tree for technical reasons. We will assume that a job's processing time is a power of $(1+\epsilon)^k$ for some integer k. This can be assumed while only loosing a $(1+\epsilon)$ extra factor in speed on each of the nodes. For jobs of size $(1+\epsilon)^i$ on a node v, we say that these jobs are in class i on v. Our algorithm will utilize the Shortest-Job-First (SJF) algorithm. This algorithm on a node v just schedules the job with the shortest *original* processing time on v amongst the jobs available to process on v. In the case of ties, the algorithm processes the oldest job in the class.

We will require a fair bit of notation in the paper. For any job J_i and node v, $p_{i,v}$ denotes the processing time of J_i on v. This may be simplified to p_i in the identical node case. The set $\mathcal{L}$ and $\mathcal{R}$ are the set of leaf nodes and nodes adjacent to the root, respectively. For any non-root node v, let $R(v)$ be the node adjacent to the root such that v is in the subtree rooted at $R(v)$. Also let $L(v)$ be the set of leaf nodes in the subtree rooted at v. The value of d_v is the total number of nodes on the path from v to $R(v)$ including v and $R(v)$. Let $\rho(v)$ be the parent of v and $c(v)$ be the children of v. For a node $v \in \mathcal{L}$ and a job J_j let $P_{v,j}$ be the sum of the processing time of job J_j on all nodes from the root to v. Note this is a lower bound on job J_j's flow time if it is assigned to leaf v.

For a given algorithm A, let $Q_v^A(t)$ be the set of jobs that have arrived by time t, have not completed processing on v, and have v in the path from the root to the leaf node they are assigned to in A. If A uses SJF on v, let $S_{v,i}^A(t)$ denote the set of jobs in $Q_v^A(t)$ which have higher priority than J_i on v and it also includes J_i. That is, the set of jobs that have smaller processing time than p_i, or that have processing time p_i and have arrived earlier than r_i. Let $d_j^A(t)$ be the remaining number of nodes J_j still requires processing time on. Let $d_{v,j}^A(t)$, for a node v that J_j still needs to be processed on, be the total number of nodes J_j needs to be processed on to reach v. Finally $p_{j,v}^A(t)$ is the remaining processing time J_j requires on node v in A at time t.

The goal of the scheduler is to minimize the total flow time of the schedule. Say an algorithm A completes job J_j at time C_j^A (it is completely processed on a leaf node). The goal is to minimize $\sum_{J_j}(C_j^A - r_j)$. This is equivalent to $\int_{t=0}^{\infty} \sum_{J_j \in \bigcup_{v\in\mathcal{L}} Q_v^A(t)} 1 dt$. However, analyzing this objective seems challenging. Due to this, we will consider a variant of fractional flow time. Fractional flow time is a standard technique used in scheduling theory. See [13] for an overview. In our variant, say that job J_j is assigned to leaf node $v_j \in \mathcal{L}$. The goal is to minimize $\int_{t=0}^{\infty} \sum_{J_j \in \bigcup_{v\in\mathcal{L}} Q_v^A(t)} \frac{p_{j,v_j}^A(t)}{p_{j,v_j}} dt$. Note that this only depends on how must a job has been processed on its *leaf* node(s). In any valid schedule each job J_j must be scheduled on a unique leaf node v_j; however, in a linear programming relaxation the job could possibly be scheduled on multiple leaf nodes. The following theorem follows immediately from known techniques. For instance, a simple extension of the proof in [25] gives the following theorem.

Theorem 3 *If an (online) algorithm A is s-speed c-competitive for fractional flow time on trees then there exists a $(1+\epsilon)s$-speed $O(c/\epsilon)$-competitive (online) algorithm A' for minimizing total flow time on trees for any constant $\epsilon > 0$. Further, if SJF is used by A on the leaves of T then one can use A as A'.*

Now we introduce the LP and the dual below which we will use throughout the paper. Let $x_{v,j,t}$ denote the amount which job J_j is scheduled on node v at time t. Let $\eta_{j,v}$ denote the total processing time job J_j requires on all nodes on the path from the root to node v. Note that for valid schedule x where each job J_j is assigned to a unique leaf node v_j, each of the two quantities, $\sum_{v\in\mathcal{L}} \sum_t x_{v,j,t} \cdot \frac{t-r_j}{p_{j,v}}$ and $\sum_{v\in\mathcal{L}} \sum_t x_{v,j,t} \cdot \eta_{j,v}/p_{j,v}$ is a lower bound to job J_j's flow time. The first quantity is a valid lower bound since $\sum_t \frac{x_{v_i,j,t}}{p_{j,v}} = 1$, and $x_{v,j,t} = 0$ for all $v \neq v_j$, and $(t - r_j)$ is at most J_j's flow time while J_j is being processed. The second quantity is a valid lower bound that incorporates the total processing time, P_{j,v_j}, J_j requires on the path to v_j if J_j is assigned to node v_j. Hence the sum of these two quantities is also a valid lower bound within a factor of two. In the objective, we also count the flow time of a job when it is finished on a root node. In general, it is sufficient to count when a job is finished on a leaf, but we will also count the root because it only increases the optimal solution by a constant factor. We use these lower bounds on the flow time of each job in the objective of the LP for technical reasons that will be useful in the dual fitting analysis. The first constraint (1) states node processes more than one job in a time step. The second constraint (2) states thata job is fully processed on the leaf nodes. Finally the last constraint (3) says that the amount a job is processed on the children of a node is at most that of the fraction it is processed on the node. Note that in an integral solution, a job cannot be processed on a child of a node until it is processed on the node.

$$\min \sum_{j\in[n]}^{n} \left(\sum_{v\in\mathcal{L}\cup\mathcal{R}} \sum_t x_{v,j,t}\Big(\frac{t-r_j}{p_{j,v}}\Big) + \sum_{v\in\mathcal{L}} x_{v,j,t}\eta_{j,v}/p_{j,v} \right) \qquad (\mathsf{LP-Primal})$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{v,j,t} \leq 1 \quad \forall v \in [m], t \qquad (1)$$

$$\sum_{v\in\mathcal{L}} \sum_{t\geq r_j} \frac{x_{v,j,t}}{p_{j,v}} \geq 1 \quad \forall j \in [n] \qquad (2)$$

$$\sum_{r_j \leq t' \leq t} \frac{x_{v,j,t'}}{p_{j,v}} \geq \sum_{r_j \leq t' \leq t} \sum_{v' \in c(v)} \frac{x_{v',j,t'}}{p_{j,v}} \quad \forall v \in [m], j \in [n], t \qquad (3)$$

$$x_{v,j,t} \geq 0 \quad \forall v \in [m], j \in [n], t \geq r_j$$

$$\max \sum_{j=1}^{n} \beta_j - \sum_{i=1}^{m} \sum_t \alpha_{v,t} \qquad (\mathsf{LP-Dual})$$

$$\text{s.t.} \quad -\alpha_{v,t} + \frac{\beta_j}{p_{j,v}} - \sum_{t'\geq t} \frac{\gamma_{\rho(v),j,t'}}{p_{j,v}} \leq \frac{t-r_j}{p_{j,v}} + \eta_{j,v}/p_{j,v}$$
$$\forall v \in \mathcal{L}, j \in [m], t \geq r_j \qquad (4)$$

$$-\alpha_{v,t} + \sum_{t'\geq t} \frac{\gamma_{v,j,t'}}{p_{j,v}} \leq \frac{t-r_j}{p_{j,v}}$$
$$\forall v \in \mathcal{R}, j \in [m], t \geq r_j \qquad (5)$$

$$-\alpha_{v,t} + \sum_{t' \geq t} \frac{\gamma_{v,j,t'}}{p_{j,v}} - \sum_{t' \geq t} \frac{\gamma_{\rho(v),j,t'}}{p_{j,v}} \leq 0$$
$$\forall v \notin \mathcal{L} \cup \mathcal{R}, j \in [n], t \geq r_j \quad (6)$$
$$\alpha_{v,t} \geq 0 \quad \forall v \in [m], t \qquad \beta_j \geq 0 \quad \forall j \in [n]$$
$$\gamma_{v,i,t} \geq 0 \quad \forall v \in [m], j \in [n], t \geq r_j$$

3. ONLINE SCHEDULING ON TREES

In this section, we show a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^6})$-competitive algorithm for the identical node case and a $(2+\epsilon)$-speed $O(\frac{1}{\epsilon^6})$-competitive algorithm for the unrelated endpoint case. Both of the results will use fractional flow time as defined in Section 2. Using Theorem 3 these results will imply Theorems 1 and 2.

3.1 Overview of the Algorithm and Analysis

In this section we give an overview of the algorithm and analysis. Say that we are given a tree T to schedule our jobs on. Our algorithm will schedule using SJF on each of the nodes of T amongst jobs which are available to schedule on that node. To us, it is somewhat surprising that such a simple greedy scheduling policy can be used on all of the nodes of the tree without considering jobs on other nodes or a job's leaf assignment. With this policy in place, the only other decision made by the algorithm is assigning jobs to leaves of the tree.

Ideally, one would like to keep the policy simple by say assigning a job to its closest leaf. Unfortunately, this will not be suitable since it does not consider the congestion in the tree. Another reasonable scheduling policy to use would be to assign the job to the leaf such that it causes the minimum increase in the objective function (assuming no more jobs arrive). Here one takes into consideration how long the new job will take to complete and how much the new job will delay already assigned jobs. Unfortunately, making such an assignment is not trivial. The main challenge comes from being unable to determine an exact algebraic expression for the cost of assigning a job to a leaf.

To circumvent this hurdle, we will prove using a potential analysis (Lemma 3) an upper bound on how long a job will wait to be completed if it is assigned to a fixed leaf node assuming no more jobs arrive. Using this bound, we would like to assign a job to the leaf such that it minimizes the increase in cost. Unfortunately, there is another hurdle. In particular, it seems challenging to analyze the algorithm on general trees. The difficulty arises due to jobs that conflict on a node, but are assigned to different leaves. These jobs maybe cause congestion a particular node, but split in different directions in the tree. For such jobs, it is challenging to determine whether or not they will actually conflict at a node or if one will be processed quickly reaching a node not shared with the other job. This results in the two jobs never interacting with each other. This comes crucially into play in the dual fitting analysis.

To overcome this second hurdle, we simplify the problem. From T we construct a new tree T', which we call a *broomstick*. Our analysis begins by showing that the objective of the optimal scheduler on the broomstick is not much larger than that for T. Then we analyze the above algorithm on trees which are broomsticks where we assign a job to the leaf that minimizes the increase in the algorithm's cost. By the structure of the broomstick, jobs will share a common path until they reach their leaf node if the are both processed by the same child of the root. This property greatly helps in setting the dual variables when determining which jobs conflict with each other.

Finally, once we have an algorithm which works well on broomsticks, we will show an algorithm for general trees. Recall that our only remaining task was to determine the leaf assignment policy for jobs. Our algorithm will construct the broomstick from T and simulate what an algorithm would have done on the broomstick. Then the algorithm will assign a job to a leaf node in T which corresponds to a leaf node on the broomstick. Finally, by construction of the broomstick, we will be able to show that the algorithm on T will have an objective smaller than that of the algorithm on the broomstick. Since we know that the broomstick algorithm has strong guarantees and that the optimal solution on T is similar to the optimal solution on the broomstick, we will have our final result. We believe that the dual fitting techniques we use for the network setting and the ideas we introduce in the reduction of the tree to broomsticks will prove useful in understanding the communication constrained scheduling problems on other network topologies.

3.2 Bounding the Waiting Time on Interior Nodes for SJF

The following lemma will prove useful throughout the analysis. The lemma essentially states that no job will be delayed by more than a constant factor multiplied by the processing it requires once it leaves the root node until it leaves its last identical node. This is a key structural property which only holds if the routers are identical; leaf nodes need not be identical. This lemma will allow us to use SJF on the nodes of the tree. It will also be useful when reducing general trees to broomstick trees and in bounding the algorithm's objective. The remainder of this section will be devoted to proving the lemma. One thing to note is that although we can bound the time on interior nodes, there is congestion at the root nodes and the endpoints if they are unrelated. Thus, we cannot simply say assign jobs to the closest leaf.

For the remainder of the section, fix any tree T in the unrelated endpoint or identical settings, a time t and any algorithm A which uses SJF amongst jobs assigned to each node. The assignment policy A uses to assign jobs to leaves can be arbitrary. Further, say that A is given s resource augmentation on all nodes except those adjacent to the root where $s \geq 1+\epsilon$ for some constant ϵ – later we will scale up the speed all nodes get uniformly. Note that we are not speeding up the nodes adjacent to the root. In the identical setting, we call all nodes of T identical. In the unrelated endpoint setting, we call the routers identical nodes and the leafs unrelated nodes. The goal of this section is to prove the following lemma.

Lemma 1 *Say that J_j is assigned to leaf node $v \in \mathcal{L}$. The total time it takes for J_j to be completed on the last identical node on its path is at most $\frac{6}{\epsilon^2} p_j d_v$ after leaving node $R(v)$.*

First we bound the volume of work which remains for jobs which are available to schedule on some identical node that is not adjacent to the root.

Lemma 2 *Consider any time t, job J_j and any identical node v which is not adjacent to the root such that J_j*

still needs to use v at time t. Then it is the case that $\sum_{J_i \in S^A_{v,j}(t) \setminus Q^A_{\rho(v)}(t)} p^A_{i,v}(t) \leq \frac{2}{\epsilon} p_j$.

PROOF. The quantity $\sum_{J_i \in S^A_{v,j}(t) \setminus Q^A_{\rho(v)}(t)} p^A_{i,v}(t)$ counts the remaining volume of work A has for jobs which have higher priority than J_j on v which are *currently* available to schedule on v. For the sake of contradiction, say the lemma is false at some time t. Let t_1 be the earliest time before time t such that node v is always processing a job that has higher priority than J_j during $(t_1, t]$. Knowing that at time t_1 A is not processing a job of higher priority than J_j on v it must be the case that $\sum_{J_i \in S^A_{v,j}(t_1) \setminus Q^A_{\rho(v)}(t_1)} p^A_{i,v}(t_1) = 0$.

Now consider the total volume of work of jobs with higher priority than J_j which can reach v during $(t_1, t]$. All of the work that arrives to v must come from jobs which pass through $\rho(v)$. Say that job J_j is of class k (e.g. $p_j = (1+\epsilon)^k$). There can be at most one job of each class partially processed on $\rho(v)$ at any point in time by definition of SJF. Besides partially processed jobs on $\rho(v)$ at time t_1, every job which reaches v during $(t_1, t]$ requires its full processing to be done on $\rho(v)$ during $(t_1, t]$. This implies that the total volume of work which can reach v from $\rho(v)$ during $(t_1, t]$ which has higher priority than J_j is at most $s(t - t_1) + \frac{2}{\epsilon} p_j$ because $\rho(v)$ has speed at most s. Now we know that v is always busy doing work on jobs which have higher priority than J_j during $(t_1, t]$ by definition of t_1 and A. Thus, v does $s(t - t_1)$ volume of work on these jobs during $(t_1, t]$. However, this implies that $\sum_{J_i \in S^A_{v,j}(t) \setminus Q^A_{\rho(v)}(t)} p^A_{i,v}(t) \leq \frac{2}{\epsilon} p_j$, a contradiction. □

Recall that $d^A_{v,i}(t)$ denotes the number of nodes on the path from where J_i is currently available to schedule to a node v. Let $P^A_i(t)$ be the remaining nodes which job J_i needs to still be processed on at time t which are identical nodes (do not include an unrelated node). We now use a potential argument to give our first upper bound of how long it will take for a job to complete processing, after passing the first node, on all remaining identical nodes.

Lemma 3 *Say A is given speed s on all nodes except those adjacent to the root for any $s \geq 1+\epsilon$. Consider job J_j which is available to schedule at time t on a node not adjacent to the root and which is not an unrelated node. Then the remaining time until either J_j is completed in the identical case, or reaches an unrelated node in the unrelated endpoint case, is a most the following assuming no jobs arrive after time t,*

$$\Phi_j(t) = \frac{1}{s} \max_{v \in P^A_j(t)} \left\{ \sum_{J_i \in S^A_{v,j}(t)} p^A_{i,v}(t) + \frac{2}{\epsilon}(d^A_j(t) - d^A_{v,j}(t)) p_j \right\}$$

PROOF. First we bound the continuous change in $\Phi_j(t)$. Let $v^* \in P_j(t)$ be the node which maximizes $\sum_{p_i \in S^A_{v,j}(t)} p^A_{i,v}(t) + \frac{2}{\epsilon}(d^A_j(t) - d^A_{v,j}(t)) p_j$. We begin by proving that there must exist a job available to schedule on v^* which is in $S^A_{v^*,j}(t)$. Indeed, say that this is not the case for the sake of contradiction. First, clearly v^* is not equal to the node job J_j is currently available on, because in this case J_j is in $S^A_{v^*,j}(t)$ and we get a contradiction. Now, it must be the case that $S^A_{v^*,j}(t) \subseteq S^A_{\rho(v^*),j}(t)$ because no job in $S^A_{v^*,j}(t)$ is available to schedule. Further, there can be at most one job of each class partially processed on node $\rho(v)$ since SJF is used on node $\rho(v)$. Thus, it must be the case that $\sum_{p_i \in S^A_{v^*,j}(t)} p^A_{i,v^*}(t) \leq \sum_{p_i \in S^A_{\rho(v^*),j}(t)} p^A_{i,\rho(v^*)}(t) + \frac{2}{\epsilon} p_j$. However, this contradicts the definition of v^* since $d^A_{v^*,j}(t) = d^A_{\rho(v^*),j}(t) + 1$. Now we know that there is a job in $S^A_{v^*,j}(t)$ which is available to schedule at time t on v^*. This implies that a job in $S^A_{v^*,j}(t)$ is worked on at time t on v^* and $\sum_{p_i \in S^A_{v^*,j}(t)} p^A_{i,v^*}(t)$ must decrease at a rate of s. Thus $\Phi_j(t)$ decreases at a rate of one.

Now we bound the discontinuous change in $\Phi_j(t)$. This occurs by job J_j moving to a new node at some time t. Say that J_j is on node v_c and moves to node v' at time t. Then $P^A_j(t)$ becomes $P^A_j(t) \setminus \{v_c\}$ and $d^A_j(t)$ decreases by one. Let v^* be the node that maximizes $\sum_{p_i \in S^A_{v,j}(t)} p^A_{i,v}(t) + \frac{2}{\epsilon}(d^A_j(t) - d^A_{v,j}(t)) p_j$ just before J_j moves to v'. If $v_c \neq v^*$ then it is easy to see that $\Phi_j(t)$ does not increase after J_j moves. Consider when $v^* = v_c$. In this case, any job in $S^A_{v',j}(t)$ that is not available to schedule at time t is also not available to schedule on v^* at time t since v^* was processing job J_j at t. These jobs contributed the same amount to $\sum_{p_i \in S^A_{v^*,j}(t)} p^A_{i,v^*}(t)$ and $\sum_{p_i \in S^A_{v',j}(t)} p^A_{i,v'}(t)$. Any other job in $S^A_{v',j}(t)$ that is available to schedule on v' at t can contribute at most $\frac{2}{\epsilon} p_j$ to $\sum_{p_i \in S^A_{v',j}(t)} p^A_{i,v'}(t)$ by Lemma 2. Thus, $\sum_{p_i \in S^A_{v^*,j}(t)} p^A_{i,v^*}(t) + \frac{2}{\epsilon} p_j \leq \sum_{p_i \in S^A_{v',j}(t)} p^A_{i,v'}(t)$ after job J_j moves. Since, $d^A_j(t)$ decreases by one and $d^A_{v',j}(t)$ decreases by one it is the case that $\frac{2}{\epsilon}(d^A_j(t) - d^A_{v^*,j}(t)) p_j$ before J_j moves is $\frac{2}{\epsilon} p_j$ more than $\frac{2}{\epsilon}(d^A_j(t) - d^A_{v',j}(t)) p_j$ after J_j moves. Thus, $\Phi_j(t)$ cannot increase after J_j moves since the term for v' after J_j moves is no more than the term for v^* before J_j moves.

Thus, knowing that $\Phi_j(t)$ will never increase so long as jobs do not arrive and the expression decreases at a rate of one at each continuous time, it must be the case that $\Phi_j(t)$ is an upper bound on the remaining time job J_j waits to be satisfied so long as no jobs arrive after time t. □

Finally we are ready to show Lemma 1.

Proof of [Lemma 1] Let r'_j be the first time when J_j is available to schedule on a node $v' \notin \mathcal{R}$. Let v_e be the last identical node on J_j's path. Let C'_j denote the time that job J_j finished on v_e. Consider Φ_j in Lemma 3. What we see is that $\Phi_j(r'_j) \leq \frac{4}{s\epsilon} d_{v_e} p_j$ by Lemma 2. Further, we see that each job J_i can contribute at most $\frac{1}{s} p_i$ to the summation in $\Phi_j(t)$ at any time t by definition of $\Phi_j(t)$. For a time $t \in (r'_j, C'_j]$ the only jobs which can contribute to $\Phi_j(t)$ which were not contributing to $\Phi_j(r'_j)$ must come from $R(v')$. Now, we know that at time r'_j there can be at most one job partially processed on $R(v')$ for each class. All of the above implies that the total increase which can happen to Φ_j during $(r'_j, C'_j]$ is $\frac{2}{s\epsilon} p_j + \frac{1}{s}(C'_j - r'_j)$ because $R(v')$ is assumed to have one speed.

Now we also know that at any point in time $t \in (r'_j, C'_j]$ it is the case that $\Phi(t)$ is non-negative and $\Phi(t)$ decreases at a rate of one. This implies that, $(C'_j - r'_j) \leq \Phi_j(r'_j) + \frac{2}{s\epsilon} p_j + \frac{1}{s}(C'_j - r'_j) \leq \frac{4}{s\epsilon} d_{v_e} p_j + \frac{2}{s\epsilon} p_j + \frac{1}{s}(C'_j - r'_j)$. The second inequality is due to the fact that $\Phi_j(r'_j) \leq \frac{4}{s\epsilon} d_{v_e} p_j$. Since $s \geq 1 + \epsilon$, we derive $C'_j - r'_j \leq \frac{6}{\epsilon^2} d_{v_e} p_j$. □

3.3 Reduction to Broomsticks

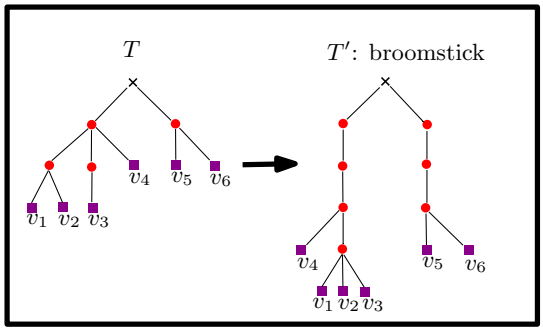

Figure 2: Tree reduction

In this section we show our reduction to broomsticks. Consider any rooted tree T in either the unrelated endpoint or identical settings and let r be the root vertex. From this tree we create a new tree T'. First add a root to T'. For every node adjacent to the root in T there is an identical node in T' adjacent to the root. Let v_0 denote some node which is a child of the root in T'. Let ℓ denote the length of the longest path from v_0 to a leaf in the subtree rooted at v_0 in T. We create a path P of length $\ell+1$ from v_0 in T' consisting of nodes $v_0, v_1, \ldots, v_\ell$. All node on this path are identical nodes. Now consider any *leaf* v which is of distance ℓ' from v_0 in T and v is in the subtree rooted at v_0. For such a node we create a node adjacent to $v_{\ell'+1}$ in P. Note the distance in T' of v to v_0 is $\ell'+2$ now, thus it has been increased by 2. In the identical setting this node is also an identical node, in the unrelated setting the processing time of a job on this node is the same as the processing time of a job on v in T. This is the reduction. See Figure 2 for a visual representation of the reduction. This theorem will let us focus on broomsticks in the dual fitting. The proof is deferred to the full version of this paper.

Theorem 4 *Let T be any tree in either the identical or unrelated endpoint settings and T' be it's corresponding broomstick. Fix any job sequence and let OPT_T denote the value of the optimal solution on T. Let $\mathrm{OPT}_{T'}$ denote the value of the optimal solution on T' where all nodes, besides those connected to the root, are given $(1+\epsilon)^2$ resource augmentation and nodes adjacent to the root are given $(1+\epsilon)$ resource augmentation for some fixed constant $\epsilon > 0$. It is the case that $\mathrm{OPT}_{T'} \leq O(\frac{1}{\epsilon^3})\mathrm{OPT}_T$.*

3.4 Assignment Policy of the Algorithm on a Broomstick

In this section we define the algorithms which we will use in the case where the tree is a broomstick. Later in Section 3.7 we will show how this algorithm can be used to generate an algorithm for general trees. In our algorithm A all of the nodes of the tree will use SJF amongst the jobs which are available to schedule on each of the nodes. Thus, it only remains to define the machine assignment policy when a job arrives. Before we define this, first we will show some bounds on how long it will take a job to be completed assuming no more jobs arrive. This will then be used to greedily decide which machine to assign a job to.

Lemma 4 *Consider any algorithm A which uses SJF on each of the identical nodes of a tree T that is a broomstick. Further say that the nodes adjacent to the root are given resource augmentation s and the nodes not adjacent to the root are given resource augmentation at least $(1+\epsilon)s$ for any $s \geq 1$ and $\epsilon > 0$. Let $v \in \mathcal{L}$ be the leaf node J_j is assigned to. Then after time t, assuming no more jobs arrive, J_j waits at most $\frac{1}{s}\sum_{J_i \in S^A_{R(v),j}(t)} p^A_{i,R(v)}(t)$ while available on $R(v)$, $\frac{6}{\epsilon^2}p_j d_v$ time steps while available on identical nodes not in $\mathcal{R}$, and $\frac{1}{s(1+\epsilon)}\sum_{J_i \in S^A_{v,j}(t)} p^A_{i,v}(t)$ while available to schedule on v.*

Proof. If J_j is available on $R(v)$ or v, then the term $\sum_{J_i \in S^A_{R(v)}(t)} p^A_{i,R(v)}(t)$ or $\sum_{J_i \in S^A_v(t)} p^A_{i,v}(t)$ decrease at a rate of at least s or $(1+\epsilon)s$, respectively. This is because the node would either process job J_j or a job of higher priority than job J_j. These terms can never increase because no job arrives. Thus, this bounds the time that job J_j can wait on while available on $R(v)$ or v. Finally, we know that J_j can wait at most $\frac{6}{\epsilon^2}p_j d_v$ time units in identical nodes which are not in $\mathcal{R}$ by Lemma 1 □

Now we are ready to define the assignment policy of our algorithm. Note that we simply need to specify the leaf node that a job should be processed on when a job arrives. Consider a job J_i which arrives at time $t = r_j$. The machine we assign a job to is the machine which minimizes the upper bound in the increase in the objective as predicted in Lemma 4. In the identical machines case we assign job J_i to the leaf node $v \in \mathcal{L}$ which minimizes the following.

$$\sum_{J_i \in S^A_{R(v),j}(t)} p^A_{i,R(v)}(t) + \frac{6}{\epsilon^2} d_v p_j + p_j \sum_{J_i \in Q^A_{R(v)}(t), p_i > p_j} 1$$

Now consider the case where endpoints are unrelated. In this case, we assign the jot to the node v such that the following is minimized.

$$\sum_{J_i \in S^A_{R(v),j}(t)} p^A_{i,R(v)}(t) + p_j \sum_{J_i \in Q^A_{R(v)}(t), p_i > p_j} 1$$
$$+ \sum_{J_i \in S^A_{v,j}(t)} p^A_{i,v}(t) + p_{j,v} \sum_{J_i \in Q^A_v(t), p_{i,v} > p_{j,v}} \frac{p^A_{i,v}(t)}{p_{i,v}} + \frac{6}{\epsilon^2} d_v p_j$$

3.5 Identical Endpoints on Broomsticks

In this section we show that our algorithm for the case of identical endpoints on a tree which is a broomstick is $O(\frac{1}{\epsilon^3})$ competitive for fractional flow time when the algorithm is given $(1+\epsilon)$ resource augmentation on nodes adjacent to the root and $(1+\epsilon)^2$ resource augmentation on the other nodes. To do this we consider a dual fitting argument. Let $F(j,v) = \sum_{J_i \in S^A_{R(v),j}(t)} p^A_{i,R(v)}(t) + \sum_{J_i \in Q^A_{R(v)}(t), p_i > p_j} p_j$ assuming $t = r_j$ and $v \in \mathcal{L}$. Note that we assign J_j to leaf node $\texttt{argmin}_{v \in \mathcal{L}}\{F(j,v) + \frac{6}{\epsilon^2} d_v p_j\}$. Let $\beta_j = F(j,v) + \frac{6}{\epsilon^2} d_v p_j$ assuming J_j is assigned to v. Let $\gamma_{v,j,t} = 0$ for all v, j and $t \neq \infty$. Let $\gamma_{v,j,\infty} = F(j,v)$. Finally, set $\alpha_{v,t} = 0$ for all v not adjacent to the root and $\alpha_{v,t} = \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t)} \frac{p^A_{i,v'}(t)}{p_{i,v'}}$ for v adjacent to the root.

The first thing to notice is that $\sum_{v,t} \alpha_{v,t}$ is exactly the factional cost for the algorithm. Also, by Lemma 4 we have that $\sum_j \beta_j$ is more than $(1+\epsilon)$ times the algorithm's cost (here we assume without loss of generality that $0 < \epsilon \leq 1/4$

since this assumption can be easily removed by scaling ϵ appropriately). Thus, the dual objective in this case is at least ϵ times the algorithm's cost. Now we will show that we can divide all the dual variables by $\frac{10}{\epsilon^2}$ to obtain a feasible solution to the dual. This will give an $O(\frac{1}{\epsilon^3})$-competitive algorithm. Our goal will be to show the following theorem which follows by scaling ϵ appropriately.

Theorem 5 *There is a $(1+\epsilon)$-speed $O(1/\epsilon^3)$-competitive algorithm for minimizing total fractional flow time on broomsticks with all identical nodes.*

To show the theorem, we show that each of the constraints are satisfied.

Lemma 5 *Constraint (4) is satisfied.*

PROOF. Fix any job J_j, any node $v \in \mathcal{L}$ and any time t. Our goal is to show that

$$\frac{\epsilon^2}{10}\left(-\alpha_{v,t} + \frac{\beta_j}{p_j} - \sum_{t' \geq t} \frac{\gamma_{\rho(v),j,t'}}{p_j}\right) - \frac{t-r_j}{p_j} - \eta_{j,v}/p_j \quad \leq \quad 0$$

Say that job J_j is assigned to node v^*. In this case, we have that $\beta_j = F(j,v^*) + \frac{6}{\epsilon^2} d_{v^*} p_j$. We also know $F(j,v) + \frac{6}{\epsilon^2} d_v p_j \geq F(j,v^*) + \frac{6}{\epsilon^2} d_{v^*} p_j$ by definition of the algorithm. Thus it suffices to show that,

$$\frac{\epsilon^2}{10}\left(-\alpha_{v,t} + \frac{F(j,v) + \frac{6}{\epsilon^2} d_v p_j}{p_j} - \sum_{t' \geq t} \frac{\gamma_{\rho(v),j,t'}}{p_j}\right) - \frac{t-r_j}{p_j} - \eta_{j,v}/p_j \leq 0,$$

which holds since $\sum_{t'>t} \gamma_{\rho(v),j,t'} = F(j,\rho(v)) = F(j,v)$ and $\eta_{j,v} = d_v p_j$. $\square$

The following lemma is the most challenging part of the dual fitting proof.

Lemma 6 *Constraint (5) is satisfied.*

PROOF. Fix any job J_j, any node $v \in \mathcal{R}$ and any time $t \geq r_j$. Our goal is to show that,

$$\frac{\epsilon^2}{10}\left(-\alpha_{v,t} + \sum_{t' \geq t} \frac{\gamma_{v,j,t'}}{p_j}\right) - \frac{t-r_j}{p_j} \leq 0$$

We know that $\alpha_{v,t} = \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t)} \frac{p^A_{i,v'}(t)}{p_{i,v'}}$. Also we have that $\sum_{t' \geq t} \gamma_{v,j,t'} = F(j,v) = \sum_{J_i \in S^A_{v,j}(r_j)} p^A_{i,v}(r_j) + \sum_{J_i \in Q^A_v(r_j), p_i > p_j} p_j$. Let V be the total volume of work that has been done on jobs in $S^A_{v,j}(r_j)$ between r_j and time t on v. Recall that since v is a node adjacent to the root, it is given $(1+\epsilon)$ resource augmentation and this implies that $V \leq (1+\epsilon)(t-r_j)$. Let V' be the total volume of jobs J_i in $Q^A_v(r_j)$ where $p_i > p_j$ that has been processed on the (only) child of v by time t. Let v_c be the child of v. This implies that $p_{i,v_c}(r_j) \geq p_j$ for $J_i \in Q^A_v(r_j)$ where $p_i > p_j$. Thus, it must be case that $V' \leq (1+\epsilon)^2(t-r_i)$. Indeed, this is the case because all jobs in $Q^A_v(r_j)$ must not have been scheduled on the node which is a child of v at all before time r_j (note there is only one child of v by definition of the broomstick). Further, all of these jobs must be eventually scheduled on this node to be completed and this node has speed $(1+\epsilon)^2$. We derive that,

$$\begin{aligned}
&\frac{\epsilon^2}{10}\left(-\alpha_{v,t} + \sum_{t' \geq t} \frac{\gamma_{v,j,t'}}{p_j}\right) - \frac{t-r_j}{p_j} \\
&= \frac{\epsilon^2}{10}\left(-\sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t)} \frac{p^A_{i,v'}(t)}{p_i} + \right. \\
&\left. \frac{\sum_{J_i \in S^A_{v,j}(r_j)} p^A_{i,v}(r_j) + \sum_{J_i \in Q^A_v(r_j), p_i > p_j} p_j}{p_j}\right) - \frac{t-r_j}{p_j} \\
&= \frac{\epsilon^2}{10}\left(-\sum_{v' \in L(v)} \sum_{J_i \in S^A_{v',j}(t)} \frac{p^A_{i,v'}(t)}{p_i}\right. \\
&- \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t) \setminus S^A_{v',j}(t)} \frac{p^A_{i,v'}(t)}{p_i} \\
&\left. + \frac{\sum_{J_i \in S^A_{v,j}(r_j)} p^A_{i,v}(r_j) + \sum_{J_i \in Q^A_v(r_j), p_i > p_j} p_i}{p_j}\right) - \frac{t-r_j}{p_j} \\
&\leq \frac{\epsilon^2}{10}\left(\frac{V}{p_j} - \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t) \setminus S^A_{v',j}(t)} \frac{p^A_{i,v'}(t)}{p_i}\right. \\
&\left. + \frac{\sum_{J_i \in Q^A_v(r_j), p_i > p_j} p_i}{p_j}\right) - \frac{t-r_j}{p_j} \qquad (7)
\end{aligned}$$

The last inequality holds since $V = \sum_{J_i \in S^A_{v,j}(r_j)} p^A_{i,v}(r_j) - \sum_{J_i \in S^A_{v,j}(r_j)} p^A_{i,v}(t)$, and for any descendent v' of v in T, $p^A_{i,v'}(t) \geq p^A_{i,v}(t)$. Before we continue to upper bound (7), we show

$$\begin{aligned}
V' &= \sum_{J_i \in Q^A_v(r_j), p_i > p_j} p^A_{i,v_c}(r_j) - \sum_{J_i \in Q^A_v(t), p_i > p_j} p^A_{i,v_c}(t) \\
&= \sum_{J_i \in Q^A_v(r_j), p_i > p_j} p_i - \sum_{J_i \in Q^A_v(t), p_i > p_j} p^A_{i,v_c}(t) \\
&\geq \sum_{J_i \in Q^A_v(r_j), p_i > p_j} p_i - \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t) \setminus S^A_{v',j}(t)} p^A_{i,v'}(t)
\end{aligned}$$

The first equality comes from the definition of V'. The second equality holds since job J_i is not completed on node v at time r_j implies that J_i has not been processed at all on its unique child node v_c. The last inequality follows from that fact that for any descendant v' of v in T, $p^A_{i,v'}(t) \geq p^A_{i,v_c}(t)$ (also note that $Q^A_{v'}(t) \setminus S^A_{v',j}(t) = Q^A_v(t), p_i > p_j$ since the algorithm is SJF).

Hence we further derive that,

$$\begin{aligned}
(7) \quad &= \quad \frac{\epsilon^2}{10}\left(\frac{V}{p_j} - \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t) \setminus S^A_{v',j}(t)} \frac{p^A_{i,v'}(t)}{p_i}\right. \\
&\qquad \left. + \sum_{J_i \in Q^A_v(r_j), p_i > p_j} \frac{p_i}{p_j}\right) - \frac{t-r_j}{p_j} \\
&\qquad \leq \frac{\epsilon^2}{10}\left(\frac{V}{p_j} + \frac{V'}{p_j}\right) - \frac{t-r_j}{p_j}
\end{aligned}$$

The last inequality follows since $V \leq (1+\epsilon)(t-r_j)$ and $V' \leq (1+\epsilon)^2(t-r_i)$, and $\epsilon \leq 1$. This completes the proof. $\square$

Lemma 7 *Constraint (6) is satisfied.*

PROOF. Fix any job J_j, any node $v \notin \mathcal{L} \cup \mathcal{R}$ and any time t. Our goal is to show that

$$\frac{\epsilon^2}{10}\left(-\alpha_{v,t} + \sum_{t' \geq t} \frac{\gamma_{v,j,t'}}{p_j} - \sum_{t' \geq t} \frac{\gamma_{\rho(v),j,t'}}{p_j}\right) \leq 0$$

We know that $\sum_{t'>t} \gamma_{\rho(v),j,t'} = F(j,\rho(v))$ and $\sum_{t'>t} \gamma_{v,j,t'} = F(j,v)$. By definition of F this implies that $\sum_{t'>t} \gamma_{\rho(v),j,t'} = \sum_{t'>t} \gamma_{v,j,t'}$. Since $\alpha_{v,t}$ is positive, the lemma follows. □

3.6 Unrelated Endpoints on Broomsticks

In this section we show that our algorithm for the case of unrelated endpoints on a tree which is a broomstick is $O(\frac{1}{\epsilon^3})$-competitive for fractional flow time when the algorithm is given $2(1+\epsilon)$ resource augmentation on nodes adjacent to the root and $2(1+\epsilon)^2$ resource augmentation on the other nodes. To do this we consider a dual fitting argument. Let $F(j,v) = \sum_{J_i \in S^A_{R(v),j}(t)} p^A_{i,R(v)}(t) + \sum_{J_i \in Q^A_{R(v)}(t), p_i > p_j} p_j$ and $F'(j,v) = \sum_{J_i \in S^A_{v,j}(t)} p^A_{i,v}(t) + p_{j,v} \sum_{J_i \in Q^A_v(t), p_i > p_j} \frac{p^A_{i,v}(t)}{p_{i,v}}$ assuming $t = r_j$ and $v \in \mathcal{L}$. We assign J_j to leaf node $\texttt{argmin}_{v \in \mathcal{L}}\{F(j,v)+F'(j,v)+\frac{6}{\epsilon^2} p_j d_v\}$. Let $\beta_j = F(j,v)+F'(j,v)+\frac{6}{\epsilon^2} d_v p_j$ assuming J_j is assigned to v. Let $\gamma_{v,j,t} = 0$ for all v, j and $t \neq \infty$. Let $\gamma_{v,j,\infty} = F(j,v)$. Finally, set $\alpha_{v,t} = 0$ for all $v \notin \mathcal{L}$ which are not adjacent to the root, $\alpha_{v,t} = \sum_{v' \in L(v)} \sum_{J_i \in Q^A_{v'}(t)} \frac{p^A_{j,v'}(t)}{p_{j,v'}}$ for v adjacent to the root and $\alpha_{v,t} = \sum_{J_i \in Q^A_v(t)} \frac{p^A_{j,v}(t)}{p_{j,v}}$ for $v \in \mathcal{L}$.

The first thing to notice that that $\sum_{v,t} \alpha_{v,t}$ is exactly twice the factional cost for the algorithm. Also, by Lemma 4 we have that $\sum_j \beta_j$ is more than $2(1+\epsilon)$ times the algorithm's cost. Thus, the dual objective in this case is at least 2ϵ times the algorithm's cost (here we without loss of generality assume that $\epsilon \leq 1/8$, and we can remove this assumption easily by scaling ϵ). Now we will show that we can divide all the dual variables by $\frac{20}{\epsilon^2}$ to obtain a feasible solution to the dual. This will give an $O(1/\epsilon^3)$ competitive algorithm. Our goal will be to show the following theorem which follows by scaling ϵ appropriately. The formal proof will appear in the full version of this paper.

Theorem 6 *There is a $(2+\epsilon)$-speed $O(1/\epsilon^3)$-competitive algorithm for minimizing total fractional flow time on broomsticks with all identical routers and unrelated endpoints.*

3.7 Algorithm for General Trees

In this section, we put all of the pieces together and give an algorithm for general trees for fractional flow time. This and the conversion from fractional flow time will complete the proofs. Our algorithm works as follows. Let T be any arbitrary tree. We will define an algorithm A_T on T. The algorithm is given resource augmentation $(1+\epsilon)$ on nodes besides those connected to the root for some constant $\epsilon > 0$. From T the algorithm creates a new tree T' which is a broomstick, preserving the resource augmentation for non-root nodes. On T' the algorithm simulates the algorithm from the previous section. Let this algorithm be denoted $A_{T'}$. Say that job J_j is assigned to node $v_j^{T'} \in \mathcal{L}$ in T' by $A_{T'}$. The algorithm A_T assigns job J_j to the node corresponding to $v_j^{T'}$ in T, which we denote by v_j^T. Then, in A_T the algorithm schedules jobs using SJF on all of the nodes.

Lemma 8 *The total flow time of A_T is at most that of $A_{T'}$.*

PROOF. To show the theorem, we will show that the flow time of each job in A_T is at most that the job waits in $A_{T'}$, which will imply the lemma. Fix any job J_j assigned to the leaf node v_j. Let d_{v_j} be the distance of v_j to a node adjacent to the root in T. First we will show that the time J_j completes on the ith identical node on its path is only sooner in A_T than in $A_{T'}$. We show this by induction on on the number of nodes which J_j has finished being processed on. For the base case, consider a node adjacent to the root. The schedule on this node is exactly the same in A_T as it is in $A_{T'}$, thus the claim holds. Now consider the ith identical node which J_j reaches. Let $v_{i,j}^T$ and $v_{i,j}^{T'}$ denote this node in T and T', respectively. By construction of T', any job that is processed on $v_{i,j}^T$ in A_T must also be processed by $v_{i,j}^{T'}$ in T (the opposite does not need to hold though). By definition of SJF and the fact that all jobs arrive to $v_{i,j}^T$ in A_T before they arrive to $v_{i,j}^{T'}$ in $A_{T'}$ by the inductive hypothesis it must be the case that J_j is complete on $v_{i,j}^T$ in A_T by the time it is completed on $v_{i,j}^{T'}$ in $A_{T'}$.

Finally, in the case that we have unrelated endpoints, we show that J_j completes on its leaf node in T by the time it completes on its leaf node in T'. From the above every job processed v_j^T in A_T is also processed on $v_j^{T'}$ in $A_{T'}$. Further, the above implies that any job assigned to v_j^T arrives to v_j^T in A_T by the time it arrives to $v_j^{T'}$ in $A_{T'}$. Thus the definition of SJF implies that J_j completes on v_j^T in A_T by the time it completes on $v_j^{T'}$ on $A_{T'}$. Thus the flow time of J_j in A_T is at most the flow time of J_j in $A_{T'}$ and we have the lemma. □

The previous lemma combined with Theorem 4 and Theorems 5 and 6 complete the proofs of Theorems 1 and 2.

4. CONCLUSION

In this paper, we initiate the study of scheduling online on a set of machines under networking constraints. As far as the authors know, this is the first time networking has been considered in conjunction with scheduling on machines in the online setting. There are many interesting open questions. One question is whether the speed required in the unrelated setting we consider can be reduced from $2+\epsilon$ to $1+\epsilon$. There appears to be a challenging hurdle in reducing the speed which arises from the processing times of jobs changing once they arrive to the machine that they are to be processed on. More broadly, it would be of interest to further address scheduling under networking constraints in general. In particular, what more general networks than those considered in this paper allow for provably good scheduling algorithms? What can be shown if jobs arrive at arbitrary nodes in the network? What can be shown for different objectives such as maximum flow time or the ℓ_k-norms of flow time? As mentioned, the work by Antoniadis et al. [5] has addressed this case when the graph is a line network in what corresponds to our identical setting when all jobs have unit size. They also showed that the objective of maximum flow time becomes hard in this setting if the network is a tree.

Acknowledgements. The first author's research was supported in part by NSF grant CCF-1409130.

5. REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, pages 63–74, 2008.

[2] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, pages 281–296, 2010.

[3] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012.

[4] M. Andrews and L. Zhang. The effects of temporary sessions on network performance. *SIAM J. Comput.*, 33(3):659–673, 2004.

[5] A. Antoniadis, N. Barcelo, D. Cole, K. Fox, B. Moseley, M. Nugent, and K. Pruhs. Packet forwarding algorithms in a line network. In *LATIN*, 2014.

[6] N. Avrahami and Y. Azar. Minimizing total flow time and total completion time with immediate dispatching. In *SPAA '03*, pages 11–18, 2003.

[7] B. Awerbuch, Y. Azar, and S. A. Plotkin. Throughput-competitive on-line routing. In *FOCS*, pages 32–40, 1993.

[8] E. Bampis, R. Giroudeau, and A. Kononov. Scheduling tasks with small communication delays for clusters of processors. *Annals OR*, 129(1-4):47–63, 2004.

[9] D. Bernstein and I. Gertner. Scheduling expressions on a pipelined processor with a maximal delay of one cycle. *ACM Trans. Program. Lang. Syst.*, 11(1):57–66, 1989.

[10] A. Borodin, J. M. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.

[11] A. Z. Broder, A. M. Frieze, and E. Upfal. A general approach to dynamic packet routing with bounded buffers. *J. ACM*, 48(2):324–349, 2001.

[12] C. Bussema and E. Torng. Greedy multiprocessor server scheduling. *Oper. Res. Lett.*, 34(4):451–458, 2006.

[13] J. S. Chadha, N. Garg, A. Kumar, and V. N. Muralidhara. A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation. In *Symposium on Theory of Computing*, pages 679–684, 2009.

[14] C. Chekuri, A. Goel, S. Khanna, and A. Kumar. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *STOC*, pages 363–372, 2004.

[15] Y.-C. Cheng and T. Robertazzi. Distributed Computation for a Tree-Network with Communication Delay. *IEEE transactions on aerospace and electronic systems*, 26(3), 1990.

[16] D. W. Engels, J. Feldman, D. R. Karger, and M. Ruhl. Parallel processor scheduling with delay constraints. In *SODA*, pages 577–585, 2001.

[17] L. Finta and Z. Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70(3):247–266, 1996.

[18] K. Fox and B. Moseley. Online scheduling on identical machines using srpt. In *SODA*, pages 120–128, 2011.

[19] N. Garg and A. Kumar. Minimizing average flow-time : Upper and lower bounds. In *FOCS*, pages 603–613, 2007.

[20] R. Giroudeau and J.-C. König. General scheduling non-approximability results in presence of hierarchical communications. *European Journal of Operational Research*, 184(2):441–457, 2008.

[21] R. Giroudeau, J.-C. König, F.-K. Moulaï, and J. Palaysi. Complexity and approximation for the precedence constrained scheduling problem with large communication delays. In *Euro-Par*, pages 252–261, 2005.

[22] A. Gupta, R. Krishnaswamy, and K. Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *WAOA*, 2012.

[23] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *NSDI*, pages 249–264, 2010.

[24] S. Im, B. Moseley, and K. Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42:83–97, June 2011.

[25] S. Im, B. Moseley, and K. Pruhs. Online scheduling with general cost functions. In *SODA*, pages 1254–1265, 2012.

[26] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[27] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[28] A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. *Algorithmica*, 43(1-2):63–80, 2005.

[29] F. T. Leighton, B. M. Maggs, and S. Rao. Packet routing and job-shop scheduling in o(congestion + dilation) steps. *Combinatorica*, 14(2):167–186, 1994.

[30] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. *J. Comput. Syst. Sci.*, 73(6):875–891, 2007.

[31] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On scheduling in map-reduce and flow-shops. In *SPAA*, pages 289–298, 2011.

[32] C. A. Phillips, C. Stein, and J. Wein. Task scheduling in networks. *SIAM J. Discrete Math.*, 10(4):573–598, 1997.

[33] K. Pruhs, J. Sgall, and E. Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. 2004.

[34] A. Srinivasan and C.-P. Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM J. Comput.*, 30(6):2051–2068, 2000.