Randomized Work-Competitive Scheduling for Cooperative Computing on k-partite Task Graphs

Chadi Kari
chadi@engr.uconn.edu

Alexander Russell
acr@cse.uconn.edu

Narasimha Shashidhar
karpoor@cse.uconn.edu

Department of Computer Science and Engineering
University of Connecticut, Storrs, CT

Abstract

A fundamental problem in distributed computing is the task of cooperatively executing a given set of t tasks by p processors where the communication medium is dynamic and subject to failures. The dynamics of the communication medium lead to groups of processors being disconnected and possibly reconnected during the entire course of the computation. The primary objective in this scenario is for the group of p processors to compute all the t tasks while minimizing the total *work* done [4]. In the partitionable network paradigm, *work* is defined as the total number of tasks performed (counting multiplicities) by all the processors during the course of the computation. In [5], the authors consider such a partitionable network scenario and analyze a simple randomized scheduling algorithm for the case where the tasks to be completed are independent of each other. In this paper, we study a natural generalization of this problem where the tasks have dependencies among them defined by a task dependency graph. In particular, we consider task dependency graphs that are k-partite task graphs. Such task dependency graphs have been studied extensively in performing dependency analysis of *PRAM* algorithms. Specifically, we present a simple randomized algorithm for p processors cooperating to perform t known tasks where the dependencies between them are defined by a k-partite task dependency graph and additionally these processors are subject to a dynamic communication medium. By virtue of the problem setting, we pursue competitive analysis where the performance of our algorithm is measured against that of the omniscient offline algorithm which has complete knowledge of the dynamics of the communication medium. We present a randomized algorithm whose competitive ratio is dependent on the dynamics of the communication medium viz. the computational width defined by [5] and also on the nature of the dependencies among the t tasks characterized by the task graph.

Key words: On-line algorithms, distributed computing, randomized algorithms, competitive analysis, partitionable networks

1 Introduction

A fundamental problem in distributed computing is the problem of cooperatively executing a given set of tasks in a dynamic setting. This problem has been studied in different settings, for instance, in message-passing models [1, 2, 4] and in partitionable network models [3, 7]. The challenge is to minimize the total *work* done and to maintain efficiency in the face of dynamically changing processor connectivity. In the partitionable network paradigm, *work* is defined as the total number of tasks performed (counting multiplicities) by all the processors during the course of the computation [4].

In this scenario, we are given a set of t tasks that must be completed in a distributed setting by a set of p processors where the communication medium is subject to failures. We assume that the t tasks are similar, in that they require the same number of computation steps to finish execution. We further assume that the tasks are idempotent - executing a task multiple times has the same effect as a single execution of the task. The tasks have a dependency relationship defined among them captured by a task dependency graph. If task B depends on task A, B cannot be performed before A.

The dynamics of the communication medium determine a processor's ability to communicate with other processors. Effectively, this partitions the processors into groups. Processors that can communicate with each other are said to belong to the same group. No communication is possible between processors in different groups. Each processor of a group is aware of all the tasks completed by the members of the group. The dynamic changes in the communication medium leads to a reconfiguration, i.e. a new partition of processors into groups. This new group of processors share knowledge of all the tasks that have been completed among them so far and then proceed to continue executing the remaining tasks from their pool of incomplete tasks until the next reconfiguration.

This processor group reconfiguration and task execution may be treated as if they were determined by an adversary. Thus, the adversary in our model performs two basic operations: reconfigures the processors into groups and also allocates the work quota for each group of processors before the next reconfiguration. The work quota is the number of tasks that can be completed by the group before the next reconfiguration takes place. While the adversary controls the *number* of tasks that a group can perform, he does not dictate *which* tasks (the identity of the tasks) the group can perform.

In this setting, the tasks have dependencies defined among them captured by a directed acyclic task graph (t-DAG) which is a k-partite task graph. Given a group of processors and the tasks known to be completed by them, an algorithm in this setting decides on the next incomplete task to be completed by this group. Each processor continues to execute tasks from the given set of t tasks until it is aware that all tasks have been completed or runs out of it's allocated work limit. Hence, given p processors and t tasks, any algorithm must execute at least $\Omega\left(t \cdot p\right)$ tasks in the scenario where all the processors are disconnected for the entire computation while any reasonable algorithm would only incur $O(t)$ *work* in the completely connected case. Hence, we treat this problem in an on-line setting and pursue competitive analysis where the performance of our algorithm is compared against that of the omniscient offline algorithm which has complete knowledge of all the future changes to the communication medium. Our setting is a generalization of the problem in [6, 5] since the tasks are no longer independent but have dependencies defined among them. We show that for this setting more pessimistic bounds hold.

2 Prior Work

Dwork, Halpern, and Waarts [4] introduced the problem of distributed cooperation for message-passing models who also defined task-oriented work. Malewicz, Russell and Shvartsman [7] introduced the notion of *h-waste* that measures the worst-case redundant work performed by h groups (or processors) when started in isolation and merged into a single group at some later time. However, all the known solutions were limited in the reconfiguration pattern of the dynamic communication medium and only addressed narrow special cases. Georgiou, Russell, and Shvartsman [5] performed competitive analysis and showed a simple randomized scheduling algorithm RS (Random Select) whose competitive ratio is tight. Their work also introduced a notion of *computation width*, which associates a natural number with a history of changes in the communication medium, and shows both upper and lower bounds on competitiveness in terms of this quantity. Specifically, they showed that their simple randomized scheduling algorithm obtains the competitive ratio $(1 + \mathbf{cw}/e)$, where $\mathbf{cw}$ is the computation width of the computation pattern determined by the dynamics of the communication medium.

3 Our Results

We follow on the work done in [5]. We study a natural generalization of the problem where the tasks to be completed are not independent of each other but have a k-partite dependency relationship defined among them. Each partition of the vertices (tasks) of the k-partite task graph is said to belong to a level. Independent tasks belong to the first level, tasks dependent on the first level tasks are at the second level and so on. The k-partite task graphs that we consider in our problem are a special kind of task graphs where every task at level l_{i+1} is dependent on every task at level $l_i,\ i = 1, \dots, k-1$ (i.e, complete set of directed edges from level l_i to level $l_{i+1},\ i = 1, \dots, k-1$). We present a simple randomized algorithm for p processors cooperating to perform t known tasks where the dependencies between them are defined by a k-partite task dependency graph with processors subject to a dynamic communication medium. We pursue competitive analysis and show that pessimistic bounds hold in this case.

Our algorithm Modified-RS extends the algorithm *Random Select*(RS) presented in [5]. Modified-RS is a simple randomized scheduling algorithm whose competitive ratio depends on the *computation width* [5] and the nature of dependencies among the tasks captured by the task graph. Since one can treat the dynamics of the communication medium (the computation pattern) as being adversarially determined, we begin by presenting an instance of a computation pattern which lower bounds the expected number of computation steps (*work*) done by any algorithm to perform t tasks on a 2-level bipartite task graph with every task at level 2 dependent on every task at level 1. We then show in section 5.3 that algorithm Modified-RS is $\left(1 + cw\left(1 - \alpha + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}}\right)\right)$-competitive for any computational (p,t)-DAG and for a 2-level task t-DAG where, cw is the **computation width** of the computational pattern (p,t)-DAG, $\alpha \in (0,1]$ denotes the fraction of tasks in the first level l_1 and $c = \frac{1}{\frac{1}{e}+o(1)}$. This competitive ratio matches the lower bound we show in section 5.1 and therefore is tight. We then extend our analysis to any k-level task t-DAG. In section 5.5 we show that Modified-RS is $\left(1 + cw\left((1-\alpha_1) + \frac{\alpha_1}{e^{\frac{\alpha_k}{\alpha_1}c^{a_k}+a_k}}\right)\right)$-competitive for any computational (p,t)-DAG and for any k-level task t-DAG where, $\alpha_i \in (0,1]$ and $c = \frac{1}{\frac{1}{e}+1}$ and where $a_i, i = 1..k$ is a sequence defined as follows, $a_1 = 1, a_{i+1} = \frac{\alpha_i}{\alpha_1}c^{a_i} + a_i$. Here, $\alpha_i \in (0,1]$ is the fraction of tasks at level $l_i,\ i = 1, \dots, k$. cw stands for the computation width of the computational (p,t)-DAG and $c_i > 0$. We also show that this result is tight as it matches the lowerbound we show in section 5.4.

When all the tasks given are independent i.e. the task t-DAG has only one level ($\alpha = 1$) the competitive ratio collapses to $(1 + cw/e)$, the bound offered by [5]. Hence, our results subsume the results of [5].

4 Model and Definitions

The problem is defined in terms of p asynchronous processors and t tasks with unique identifiers, initially known to all processors. For our purposes the tasks are idempotent and similar, i.e., each task requires the same number of computation steps.

Definition 1. *A t-DAG is a directed acyclic k-partite graph $G = (V, E)$, where $V = \dot{\bigcup}_{l=1}^{k} V_l = [t] = \{1 \dots t\}$. Edge $e = (t_i^l, t_j^{l+1}) \in E,\ l = 1, \dots, k-1,\ i \neq j$ if and only if task t_j^{l+1} depends on task t_i^l. We write $t_i^l < t_j^{l+1}$ if task t_j^{l+1} depends on task t_i^l. Here, $\dot{\bigcup}$ stands for disjoint union.*

We only consider task graphs where a task on level l_{i+1} depends on all tasks of level l_i. The computation pattern i.e., the computational (p,t)-DAG defined below captures the behavior of the adversary that determines both the partitioning and the number of tasks allocated to each group of the partition.

Definition 2. *A computational (p,t)-**DAG** is a directed acyclic graph $C = (V, E)$ augmented with a weight function $h : V \to [t] \cup \{0\}$ and a labeling $g : V \to 2^{[p]} \setminus \{\emptyset\}$ so that:*

- *For any maximal path $P = (v_1, v_2, \dots, v_k)$ in C, $\sum_{i=1}^{k} h(v_i) \geq t$. (This guarantees that any algorithm terminates during the computation described by the DAG.)*

- *g possesses the following "initial conditions":*

$$[p] = \dot{\bigcup_{v:\ in(v)=0}} g(v).$$

- *g respects the following "conservation law":*
  There is a function $\phi : E \to 2^{[p]} \backslash \{\emptyset\}$ so that for each $v \in V$ with $in(v) > 0$,

$$g(v) = \dot{\bigcup_{(u,v)\in E}} \phi((u,v)),$$

  and for each $v \in V$ with $out(v) > 0$,

$$g(v) = \dot{\bigcup_{(v,u)\in E}} \phi((v,u)).$$

In the above definition, $in(v)$ and $out(v)$ denote the in-degree and out-degree of v respectively. Finally, for the two vertices $u, v \in V$, we write $u \le v$ if there is a directed path from u to v; we then write $u < v$ if $u \le v$ and u and v are distinct.

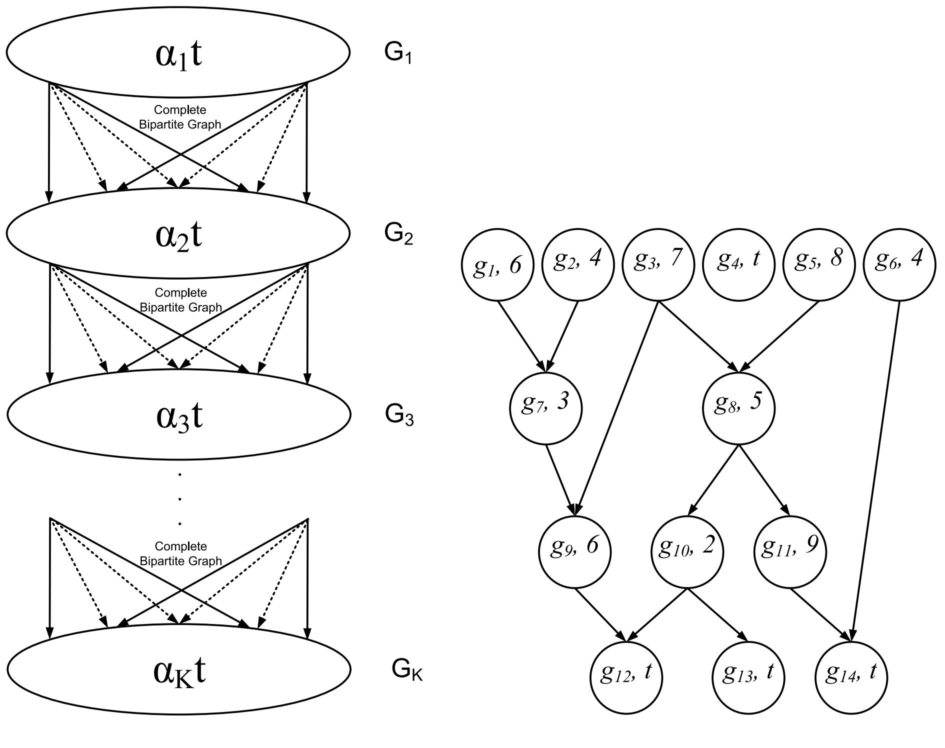

Figure 1: Right: An example of a computational $(15, t)$-DAG, Left: task t-DAG, $\alpha_{1,2,\ldots,k} \in (0, 1]$.

Example. As an example, consider the computational $(15, t)$-DAG shown on Figure 1. Here we have $g_1 = \{p_1, p_2, p_3\}$, $g_2 = \{p_4, p_5\}$, $g_3 = \{p_6, p_7\}$, $g_4 = \{p_8, p_9\}$, $g_5 = \{p_{10}, p_{11}\}$, $g_6 = \{p_{12}, p_{13}, p_{14}, p_{15}\}$, $g_7 = \{p_1, p_2, p_3, p_4, p_5\}$, $g_8 = \{p_7, p_{10}, p_{11}\}$, $g_9 = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, $g_{10} = \{p_{10}, p_{11}\}$, $g_{11} = \{p_7\}$, $g_{12} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_{11}\}$, $g_{13} = \{p_{10}\}$ and $g_{14} = \{p_7, p_{12}, p_{13}, p_{14}, p_{15}\}$.

Brief Description of the example: This computation pattern models a dynamic communication medium with the following characteristics.

- The first reconfiguration occurs when groups g_1 and g_2 merge to form group g_7. The members of group g_7 is the union of the processors in groups g_1 and g_2. Prior to this reconfiguration groups g_1 and g_2 performed exactly 6 and 4 units of work respectively.
- Group $g_4 = \{p_8, p_9\}$ runs in isolation for the entire computation and hence does t units of work.
- Group $g_3 = \{p_6, p_7\}$ splits into groups g_8 and g_9 and performs 7 units of work before the reconfiguration. Group g_9 is a result of a merge by groups g_7 and processor p_6 of group g_3. Similarly, Group g_8 is the result of a merge by groups g_5 and processor p_5 of group g_3.
- Group g_8 performs 5 units of work before splitting into two groups, g_{10} and g_{11} which proceed to perform 2 and 9 units of work respectively before the next reconfiguration (assuming that there are at least 2 or 9 tasks remaining respectively, otherwise they would have performed the remaining tasks)
- Finally, Group g_{12} is a result of a merge by groups g_9 and processor p_{11} of group g_{10}. Similarly, Group g_{13} contains the processor p_{10} of group g_{10}. Group g_{14} is the result of a merge by groups g_{11} and g_6.
- The processors in g_{12}, g_{13} and g_{14} run until completion with no further reconfigurations.

Definition 3. *Given a computational DAG $C = (V, E)$ and a vertex $v \in V$, we define the **predecessor graph** at v, denoted $P_C(v)$, to be the subgraph of C that is formed by the union of all paths in C terminating at v. Likewise, the **successor graph** at v, denoted $S_C(v)$, is the subgraph of C that is formed by the union of all the paths in C originating at v.*

Associated with any directed acyclic graph (DAG) $C = (V, E)$ is the natural **vertex poset** $(V, \leq)$ where $u \leq v$ if and only if there is a directed path from u to v. Then the **width of** C, denoted $\mathbf{w}(C)$, is the width of the poset $(V, \leq)$.

Definition 4. *The **computation width** of a computational DAG $C = (V, E)$, denoted $cw(C)$, is defined as $cw(C) = \max_{v \in V} \mathbf{w}(S(v))$.*

Let OPT denote the optimal (off-line) algorithm. $W_{\text{OPT}}(C)$ and $W_R(C)$ is the *work* done by the optimal algorithm and a randomized algorithm R.

We treat randomized algorithms as distributions over deterministic algorithms; for a set Ω and a family of deterministic algorithms $\{D_r \mid r \in \Omega\}$ we let $R = \mathfrak{R}(\{D_r \mid r \in \Omega\})$ denote the randomized algorithm where r is selected uniformly at random from Ω and scheduling is done according to D_r. For a real-valued random variable X, we let $\mathbb{E}[X]$ denote its expected value. We let OPT denote the optimal (off-line) algorithm. Specifically, for each C we define $W_{\text{OPT}}(C) = \min_D W_D(C)$.

Definition 5. [9, 10] *Let γ be a real valued function defined on the set of all (p, t)-DAGs (for all p and t). A randomized algorithm R is γ-**competitive** if for all computation patterns C,*

$$\mathbb{E}[W_{D_r}(C)] \leq \gamma(C) W_{OPT}(C),$$

this expectation being taken over uniform choice of $r \in \Omega$.

5 Lower bounds and Algorithm Modified-RS

In this section we give a lower bound on our problem for 2-level task graphs and we present the algorithm Modified-RS. We then show that for 2-level task graphs the competitive ratio of Modified-RS is tight.

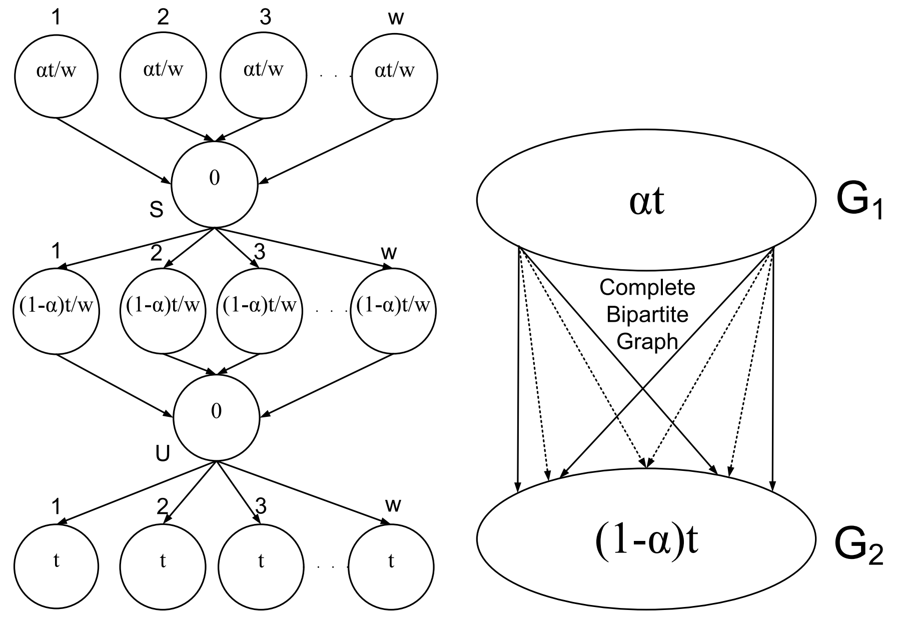

Figure 2: Left: computational (p, t)-DAG, Right: task t-DAG, $\alpha \in (0, 1]$.

5.1 A lower bound for 2-level task graphs

Theorem 1. *Let A be a scheduling algorithm for 2-level task graphs, α be the fraction of tasks at level l_1. Then,*

$$W_A \geq \left(1 + cw\left((1-\alpha) + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}e+1}}\right)\right) W_{OPT}$$

Proof. Consider the computation pattern given by the computational (p, t)-DAG and the task t-DAG in Figure 2. Initially, the computation pattern has w groups each consisting of a single processor. Let $t >> w$ and $t \mod w = 0$. Each processor completes $\frac{\alpha t}{w}$ tasks before they are merged and allowed to exchange information about completed tasks before being split again into w processors where each processor is allowed to complete $\frac{(1-\alpha)t}{w}$ tasks, at this point they are merged again and then split into w processors. For this computation pattern the optimal off-line algorithm completes all the t tasks at the formation of the group $g(U)$ and accrues exactly t work. Let $P_i \subset G_1$ denote the set of $\frac{\alpha t}{w}$ tasks for processor i. We analyze A when the tuple $P = (P_1, \ldots, P_w)$ is selected uniformly at random among all such tuples. We will show that for any algorithm A there is a configuration of the P_i such that

$$W_A \geq \left(1 + (1-o(1))cw\left((1-\alpha) + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}e+1}}\right)\right) t$$

For a fixed $\tau \in G_1$, let $p_\tau = Pr[\tau \notin P_1]$ then $Pr[\tau \notin \bigcup_i P_i] = p_\tau^w$. Let L_S be the random variable whose value is the number of tasks of G_1 left undone at the formation of group $g(S)$.

$$\mathbb{E}[|L_S|] = \mathbb{E}[|[t] - \bigcup_i P_i|] = \sum_\tau p_\tau^w$$

The function $x \to x^k$ is convex on the interval $[0, \infty)$, so, $\sum_\tau p_\tau^k$ is minimized when the p_τ are equal. Now,

$$\frac{\alpha t}{w} = \mathbb{E}[|P_1|] = \sum_\tau Pr[\tau \in P_1] = \sum_{\tau \in G_1} (1 - p_\tau)$$

So, $\sum_{\tau \in G_1} p_\tau = \alpha t - \frac{\alpha t}{w} = \alpha t \left(1 - \frac{1}{w}\right)$ and hence,

$$\mathbb{E}[|L_S|] \geq \alpha t \left(1 - \frac{1}{w}\right)^w$$

Let T be the actual number of tasks left undone at the formation of $g(S)$. In lemma 2, we show that with a high probability $T \geq \alpha t \left(1 - \frac{1}{w}\right)^w (1 - o(1))$.
Clearly, at the formation of group $g(U)$, OPT, the optimal off-line algorithm would have finished executing all the t tasks. Let α_0 be such that $\frac{(1-\alpha_0)t}{w} = T$, by choosing $\alpha > \alpha_0$ we have that after the first merge and reconfiguration the number of tasks not completed from G_1 is greater that $\frac{(1-\alpha)t}{w}$ and thus the sets of tasks that algorithm A chooses for the w processors after $g(S)$ have to be picked from G_1.
For a specific task τ in G_1 not completed at $g(S)$ (i.e. $\tau \in [T]$), let $p_\tau = Pr[\tau \notin P_1]$ then $Pr[\tau \notin \bigcup_i P_i] = p_\tau^w$.
Let L_U be the random variable whose value is the number of tasks of G_1 left undone at the formation of group $g(U)$.

$$\mathbb{E}[|L_U|] = \mathbb{E}[|[T] - \bigcup_i P_i|] = \sum_{\tau \in [T]} p_\tau^w$$

As before, the function $x \to x^k$ is convex on the interval $[0, \infty)$, so $\sum_\tau p_\tau^k$ is minimized when the p_τ are equal. Now,

$$\frac{(1-\alpha)\,t}{w} = \mathbb{E}[|P_1|] = \sum_{\tau \in [T]} Pr[\tau \in P_1] = \sum_{\tau \in [T]} (1 - p_\tau)$$

So, $\sum_{\tau \in [T]} p_\tau = T - \frac{(1-\alpha)t}{w} = T\left(1 - \frac{(1-\alpha)t}{wT}\right)$ and hence,

$$\begin{aligned}
\mathbb{E}[|L_U|] &\geq T\left(1 - \frac{(1-\alpha)t}{wT}\right)^w \\
&\geq \alpha t \left(1 - \frac{1}{w}\right)^w (1 - o(1)) \left(1 - \frac{(1-\alpha)t}{\alpha t \left(1 - \frac{1}{w}\right)^w (1 - o(1))}\right)^w
\end{aligned}$$

As $lim_{w \to \infty} \left(1 - \frac{1}{w}\right)^w = \frac{1}{e}$ and choosing $\alpha > \alpha_0 = \frac{e}{w+e}$ we get,

$$\mathbb{E}[|L_U|] \quad \geq \quad (1 - o(1))\frac{\alpha t}{e} \frac{1}{e^{\frac{1-\alpha}{\alpha} e (1 - o(1))}} \tag{1}$$

In particular there must exist selection of the P_i which achieves this bound. Note that after $g(U)$ the processors are split again into w processors where they will complete the remaining $(1 - o(1))\frac{\alpha t}{e}\frac{1}{e^{\frac{1-\alpha}{\alpha}e(1-o(1))}}$ tasks of G_1 and the $(1 - \alpha)t$ tasks of G_2.
So finally the total work of A is at least:

$$\left(1 + (1 - o(1))cw\left((1 - \alpha) + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}e+1}}\right)\right)t$$

□

Note that when the tasks are independent ($\alpha = 1$) the lower bound is $1 + (1 - o(1))\frac{cw}{e}$ which matches the result of [5] but the lower bound gets more pessimistic as the fraction of independent tasks gets smaller.

Lemma 2. $Pr[|T - \mathbb{E}[L_S]| \geq 4log(t)\sqrt{\alpha t}] \leq \frac{1}{O(t^2)}$

Proof. Let S be the set of tasks randomly picked by the processors in figure 2 at the formation of the group $g(S)$. For $1 \le i \le \alpha t$ and $1 \le j \le \frac{\alpha t}{w}$ we define the random variable Y_j^i that takes the value of the j^{th} task picked by the processor $\lceil \frac{i}{\frac{\alpha t}{w}} \rceil$. Note that in figure 2 each processor picks $\frac{\alpha t}{w}$ tasks. Let the function f be the cardinality of the set S and let $X_0, \ldots, X_{\alpha t}$ be the sequence of random variables such that $X_i = \mathbb{E}[f(S)|Y_1^1, \ldots, Y_i^l]$ with $l = \lceil \frac{i}{\frac{\alpha t}{w}} \rceil$, $X_0 = \mathbb{E}[f(S)]$ and $X_{\alpha t} = f(S)$. This sequence of random variables is a martingale and we can use Azuma's inequality to derive the error probability bound. $Pr[|X_{\alpha t} - X_0| \ge \lambda\sqrt{\alpha t}] \le 2e^{\frac{-\lambda^2}{2}}$. L_S is the random variable whose value is the number of tasks of G_1 left undone at the formation of group $g(S)$, so $\mathbb{E}[L_S] = \alpha t - X_0$ and $T = \alpha t - X_{\alpha t}$. Thus for $\lambda = 4\log(t)$

$$Pr[|T - \mathbb{E}[L_S]| \ge 4log(t)\sqrt{\alpha t}] \le \frac{1}{O(t^2)}$$

□

5.2 Description of Modified-RS

We pre-process our task graph to label its nodes with the labeling procedure below. This labeling procedure assigns every task at level l_i, the label i. Tasks with no dependencies at level l_1 get label 0. A secondary result of this labeling procedure is that it transforms any arbitrary task graph into a k-level task graph suitable for our analysis. Modified-RS and its analysis are described formally in the following section.

Given a task t-DAG $G = (V, E)$ we describe the labeling procedure $l : V \to \mathbb{N}$ that assigns a label to every vertex in the following manner:

- $\forall v \in V$ s.t. $in(v) = 0$, $l(v) = 0$, i.e., the independent tasks have label 0.
- Starting with u s.t. $l(u) = 0$, recursively label the remaining tasks in G in the following manner: if $(u, v) \in E$, $l(v) = l(u) + 1$. If v has already a label i, overwrite i with the new label j only if $j > i$.

Modified-RS

We are now ready to define Modified-RS (m-RS) where a processor with knowledge that tasks in a set $K \subset V$ have been completed chooses the next task τ to be completed at random from $V \setminus K$ if and only if $\forall t \in V \setminus K$, $l(\tau) \le l(t)$.

5.3 Analysis of Modified-RS

In this section, we analyze the competitive ratio of Modified-RS and we show it's tight by obtaining the upper bound of the work performed by our algorithm on any computation pattern (p, t)-DAG and a 2-level task t-DAG which matches the lower bound of the previous section.

5.3.1 Upper Bound for m-RS on a 2-level task DAG

We start by defining saturated and unsaturated vertices.

Definition 6. *Let C be a computational (p, t)-DAG. Associated with C are the two functions $h : V \to [t] \cup \{0\}$ and $g : V \to 2^{[p]} \setminus \{\emptyset\}$. For a subgraph $C' = (V', E')$ of C, let $H(C') = \sum_{v \in V'} h(v)$. Then, we say that a vertex $v \in V$ is* saturated *if $H(P_C(v)) \le t$; otherwise, v is* unsaturated. *Let $\mathcal{S}$ denote the set of saturated vertices and $\mathcal{U}$ the set of unsaturated vertices . Note that if v is saturated, then the group $g(v)$ must complete $h(v)$ tasks* regardless of the scheduling algorithm used. *Along these same lines, if v is an unsaturated vertex for which $t > \sum_{u<v} h(u)$, the group $g(v)$ must complete at least* $\max(h(v), t - \sum_{u<v} h(u))$ *tasks under any*

scheduling algorithm. A saturated vertex s is l_1-unsaturated if $H(P_C(s)) \geq \alpha t$. If v is an unsaturated vertex for which $\sum_{u<v} h(u) < t$ we replace v by a pair of vertices v_s and v_u and an edge (v_s, v_u) such that all edges directed into v get directed into v_s and all edges directed out of v get directed out v_u and h is redefined so that $h(v_s) = t - \sum_{u<v} h(u)$ and $h(v_u) = h(v) - h(v_s)$. Doing this will allow us to have

$$v \quad \textit{unsaturated} \Rightarrow \sum_{u<v} h(u) \geq t$$

In the same way we can have

$$v \quad l_1 - \textit{unsaturated} \Rightarrow \sum_{u<v} h(u) \geq \alpha t$$

We will also use a generalized degree-counting argument shown is [5]

Lemma 3. *Let $G = (U, V, E)$ be an undirected bipartite graph. for a vertex v, let $\Gamma(v)$ be the set of vertices adjacent to v. Suppose for some $A > 0$ and for each $u \in U$ we have $\sum_{v \in \Gamma(u)} h(v) \leq A$ and that for some $B > 0$ each vertex $v \in V$ we have $\sum_{u \in \Gamma(v)} h(u) \geq B$ then*

$$\frac{\sum_{u \in U} h(u)}{\sum_{v \in V} h(v)} \geq \frac{B}{A}$$

Theorem 4. *Algorithm Modified-RS is $\left(1 + cw\left(1 - \alpha + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}}\right)\right)$-competitive for any computational (p,t)-DAG and for a 2-level task t-DAG. Here, cw stands for the* ***computation width*** *of the computational (p,t)-DAG, $\alpha \in (0, 1]$ (α is the fraction of tasks at level l_1) and $c = \frac{1}{\frac{1}{e}+o(1)}$.*

Proof. By the definition of an unsaturated/saturated vertex we have $W_{OPT} \geq \sum_{s \in \mathcal{S}} h(s)$ and

$$\sum_{u<v} h(u) \geq t \tag{2}$$

We define T_v as the random variable denoting the number of tasks that m-RS completes at vertex v (if v is saturated then $T_v = h(v)$).
Given the (p,t)-DAG $C = (V, E)$ construct the following bipartite graph $G = (\mathcal{S}, \mathcal{U}, E(G))$ s.t $E(G) = \{(s, u) | s < u\}$. Assign the weight $\mathbb{E}[T_v]$ to vertex v. By equation 2

$$\forall u \in \mathcal{U} \sum_{s \in \Gamma(u)} \mathbb{E}[T_s] = \sum_{s \in \Gamma(u)} h(s) \geq t \tag{3}$$

We show that $\forall s \in \mathcal{S}$,

$$\forall u \in \mathcal{U} \sum_{u \in \Gamma(s)} \mathbb{E}[T_u] \leq cw\left(1 - \alpha + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}}\right) t \tag{4}$$

$$W_{m-RS} = \mathbb{E}\left[\sum_v T_v\right] = \mathbb{E}\left[\sum_{s \in \mathcal{S}} T_s\right] + \mathbb{E}\left[\sum_{u \in \mathcal{U}} T_u\right]$$

By linearity of expectation

$$W_{m-RS} = \sum_{s \in \mathcal{S}} \mathbb{E}[T_s] + \sum_{u \in \mathcal{U}} \mathbb{E}[T_u]$$

Note that equations 4 and 3 together with lemma 3 gives

$$\begin{aligned} W_{m-RS} &\leq \left(1 + cw\left(1 - \alpha + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}}\right)\right) \sum_{s \in \mathcal{S}} \mathbb{E}[T_s] \\ &= \left(1 + cw\left(1 - \alpha + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}}\right)\right) \sum_{s \in \mathcal{S}} h(s) \\ &\leq \left(1 + cw\left(1 - \alpha + \frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}}\right)\right) W_{OPT} \end{aligned}$$

as desired.

Now we show equation 4. Consider $s \in \mathcal{S}$ a saturated vertex and it's successor graph $S(s)$. $S(s)$ is covered by w paths $P_i, i = 1 \dots w$ where w is at most cw. For each path P_i let u_i^0 be the first unsaturated vertex and s_i^1 be the first l_1-unsaturated vertex. Let $L_{s_i^1}$ be the random variable whose value is the set of tasks left incomplete by m-RS at the formation of the group $g(s_i^1)$. For a fixed $\tau \in G_1$ conditioned upon the event that τ is not yet complete, the probability that τ is *not* chosen by m-RS at a selection point in $P_C(s_i^1)$ is no more than $\left(1 - \frac{1}{\alpha t}\right)$. As s_i^1 is l_1-unsaturated $\sum_{v<s_i^1} h(v) \geq \alpha t$ then for each i,

$$Pr[\tau \in L_{s_i^1}] \leq \left(1 - \frac{1}{\alpha t}\right)^{\alpha t} \leq \frac{1}{e}$$

As there are αt tasks in G_1, $\mathbb{E}[|L_{s_i^0}|] \leq \frac{\alpha t}{e}$. As before, we let T be the actual number of tasks left undone at vertex S. By the same reasoning as in lemma 2, we see that,

$$Pr[|T - \mathbb{E}[L_{s_i^1}]| \geq 4log(t)\sqrt{\alpha t}] \leq \frac{1}{O(t^2)}$$

Then $T \leq \alpha t \left(\frac{1}{e} + o(1)\right) = \frac{\alpha t}{c}$ with high probability. Now for each u_i^0 consider the subgraph H_i of the computational (p,t)-DAG defined as $H_i = \left(\bigcup_{j=1}^{w} S_C(s_j^1)\right) \bigcap P_C(u_i^0)$. Given a task $\tau \in G_1$ conditioned upon the event that τ is not yet complete, the probability that τ is *not* chosen by m-RS at a selection point in ${H_i}$[1] is no more than $(1 - \frac{c}{\alpha t})$. As $\sum_{\{s_i^1<v\}\cap\{v<u_i^0\}} h(v) \geq (1-\alpha)t$, for each i

$$Pr[\tau \in L_{u_i^0}] \leq \left(1 - \frac{c}{\alpha t}\right)^{(1-\alpha)t} \leq \frac{1}{e^{\frac{1-\alpha}{\alpha}c}}.$$

So the $\mathbb{E}[|L_{u_i^0}|] \leq (1-\alpha)t + \frac{\alpha t}{e^{\frac{1-\alpha}{\alpha}c+1}}$ Let X_i be the random variable whose value is the number of tasks done by m-RS on the portion of P_i consisting of unsaturated vertices. $X_i \leq |L_{u_i^0}|$ so $\mathbb{E}[X_i] \leq \mathbb{E}[|L_{u_i^0}|]$. By linearity of expectation

$$\mathbb{E}[\sum_i X_i] \leq cw((1-\alpha)t + \frac{\alpha t}{e^{\frac{1-\alpha}{\alpha}c+1}})$$

Now every unsaturated vertex appears in some P_i in $S_C(s)$ hence,

$$\sum_{u\in\Gamma(s)} \mathbb{E}[T_u] \leq \mathbb{E}[\sum_i X_i] \leq cw(1-\alpha+\frac{\alpha}{e^{\frac{1-\alpha}{\alpha}c+1}})t$$

□

In the next section we extend the lower bound and upper bound results to any k-level task t-DAG.

[1]If $\exists s_i^1 s.t. \sum_{s_i^1<v<u_i^0} h(v) < \frac{\alpha t}{c}$), the competitive ratio is no worse than $1 + cw(1-\alpha)$.

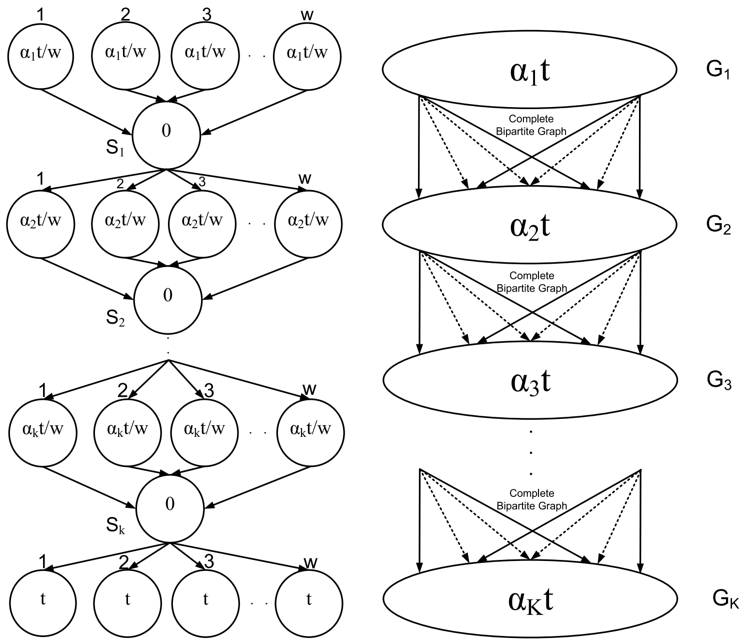

Figure 3: Right: task t-DAG, $\alpha_{1,2,...,k} \in (0, 1]$, Left: An instance of a k-level computational (p, t)-DAG

5.4 A Lower bound for k-level task graphs

Theorem 5. *Let A be a scheduling algorithm for k-level task graphs. Then,*

$$W_A \geq \left(1 + cw\left((1-\alpha_1) + \frac{\alpha_1}{e^{(1-o(1))\frac{\alpha_k}{\alpha_1}e^{a_k}+a_k}}\right)\right) W_{OPT}$$

where $a_i, i = 1..k-1$ is a sequence defined as follows,
$a_1 = 1$
$a_{i+1} = (1 - o(1))\frac{\alpha_i}{\alpha_1}e^{a_i} + a_i$

Proof. Consider the computation pattern given by the computational (p, t)-DAG and the task t-DAG in Figure 1. Let $t >> w$ and $t \mod w = 0$. Initially, the computation pattern has w processors. For this computation pattern the optimal off-line algorithm completes all the t tasks at the formation of the group $g(S_k)$ and accrues exactly t work. Let $P_j \subset G_1$ denote the set of $\frac{\alpha_i t}{w}$ tasks for processor j at level l_i of the computation pattern (i.e. the w processors resulting from the split of group $g(S_{i-1})$) . We analyze A when the tuple $P = (P_1, \ldots, P_w)$ is selected uniformly at random among all such tuples. We will show that for any algorithm A there is a configuration of the P_i such that

$$W_A \geq \left(1 + cw\left((1-\alpha_1) + \frac{\alpha_1}{e^{(1-o(1))\frac{\alpha_k}{\alpha_1}e^{a_k}+a_k}}\right)\right) t$$

We first show by induction that the expected number of tasks left undone by A at the formation of group $g(S_k)$ is

$$E[|L_{S_k}|] \geq \frac{\alpha_1 t}{e^{(1-o(1))\frac{\alpha_k}{\alpha_1}e^{a_k}+a_k}}$$

when $lim_{w\to\infty}\left(1 - \frac{1}{w}\right)^w = \frac{1}{e}$. This will give us the bound on W_A.

The base case is shown in theorem 1. So we assume the result for $i-1$, namely

$$E[|L_{S_{i-1}}|] \geq \frac{\alpha_1 t}{e^{(1-o(1))\frac{\alpha_{i-1}}{\alpha_1}e^{a_{i-1}}+a_{i-1}}}$$

Let T_{i-1} be the actual number of tasks left undone at the formation of $g(S_{i-1})$. In lemma 6, we show that with a high probability $T_{i-1} \geq E[|L_{S_{i-1}}|](1-o(1))$. As in the proof of theorem 1, for some $\alpha_i i = 1, \dots i-1$ the number of tasks left undone from G_1 at $g(S_{i-1})$ can exceed the $\frac{\alpha_i t}{w}$ tasks that the w processors at level l_i can complete.

For a specific task τ in G_1 not completed at $g(S_i)$ (i.e. $\tau \in [T_{S_{i-1}}]$), let $p_\tau = Pr[\tau \notin P_1]$ then $Pr[\tau \notin \bigcup_i P_i] = p_\tau^w$. Let L_{S_i} be the random variable whose value is the number of tasks of G_1 left undone at the formation of group $g(S_i)$.

$$\mathbb{E}[|L_{S_i}|] = \mathbb{E}[|T_{S_{i-1}}|] - \bigcup_i P_i|] = \sum_{\tau \in [T_{S_{i-1}}]} p_\tau^w$$

As before, the function $x \to x^k$ is convex on the interval $[0, \infty)$, so $\sum_\tau p_\tau^k$ is minimized when the p_τ are equal. Now,

$$\frac{\alpha_i t}{w} = \mathbb{E}[|P_1|] = \sum_{\tau \in [T_{S_{i-1}}]} Pr[\tau \in P_1] = \sum_{\tau \in [T_{S_{i-1}}]} (1-p_\tau)$$

So, $\sum_{\tau \in [T_{S_{i-1}}]} p_\tau = T_{S_{i-1}} - \frac{(1-\alpha_i)t}{w} = T_{S_{i-1}}\left(1 - \frac{(1-\alpha_i)t}{wT_{S_{i-1}}}\right)$ and hence,

$$\mathbb{E}[|L_{S_i}|] \quad \geq \quad T\left(1 - \frac{(1-\alpha_i)t}{wT_{S_{i-1}}}\right)^w$$

As $lim_{w\to\infty}\left(1-\frac{1}{w}\right)^w = \frac{1}{e}$, we have

$$E[|L_{S_i}|] \geq \frac{\alpha_1 t}{e^{(1-o(1))\frac{\alpha_i}{\alpha_1}e^{a_i}+a_i}}$$

and our induction is complete.

□

Lemma 6. *$Pr[|T - \mathbb{E}[L_g(S_i)]| \geq 4log(t)\sqrt{\alpha_i t}] \leq \frac{1}{O(t^2)}$, for $i = 1 \dots k$*

Proof. The proof follows exactly the proof of lemma 2 by replacing S by S_i and α by α_i □

5.5 Upper Bound for m-RS on a k-level task DAG

Theorem 7. *Algorithm Modified-RS is $\left(1 + cw\left((1-\alpha_1) + \frac{\alpha_1}{e^{\frac{\alpha_k}{\alpha_1}c^{a_k}+a_k}}\right)\right)$-competitive for any computational (p,t)-DAG and for any k-level task t-DAG where, $\alpha_i \in (0,1]$ and $c = \frac{1}{\frac{1}{e}+1}$ and where $a_i, i = 1..k$ is a sequence defined as follows,*
$a_1 = 1$
$a_{i+1} = \frac{\alpha_i}{\alpha_1}c^{a_i} + a_i$

Proof. A vertex v is l_i-unsaturated if $\sum_{u<v} h(v) \geq \alpha_i t$. We will proceed as in the proof of theorem 4, since the reasoning is the same we only need to show that $\sum_{u \in \Gamma(s)} \mathbb{E}[T_u] \leq cw\left((1-\alpha_1) + \frac{\alpha_1}{e^{\frac{\alpha_k}{\alpha_1}c^{a_k}+a_k}}\right)t$. Let s_i^j be the first l_j-unsaturated vertex on Path P_i, We will proceed by showing the theorem by induction on the tasks left undone at the formation of the group $g(s_i^j)$. For $j = 1$ the result is show in theorem 4. Assume

at the formation of group $g(s_i^{j-1}), T_{j-1} \leq \frac{\alpha_1 t}{c^{\frac{\alpha_{j-1}}{\alpha_1}} c^{\alpha_{j-1}+a_{j-1}}}$ we will show that at s_j our induction hypothesis holds. Now as in Section 5.2 consider for each unsaturated vertex u_i^0 the subgraph

$$H_{ij} = \left(\bigcup_k S_C(s_k^j) \right) \bigcap P_C \left(s_i^{j+1} \right),$$

Given a task $\tau \in G_1$ not yet complete at the formation of the group $g(s_i^j)$ the probability that τ is *not* chosen by Modified-RS at the formation of group $g(s_i^j)$ is [2] As $\sum_{s_i^j < v < u_i^0} h(v) \geq \alpha_j)t$, for each i

$$Pr[\tau \in L_{s_i^j}] \leq \left(1 - \frac{\alpha_j}{\alpha_j T_{j-1}}\right)^{(\alpha_j)t} \leq \frac{1}{e^{\frac{\alpha_j}{T_{j-1}}}}.$$

and the expected value of tasks left undone at s_i^j is less than $\leq \frac{\alpha_1 t}{c^{\frac{\alpha_j}{\alpha_1}} c^{\alpha_j + a_j}}$ and thus result follows.

$\square$

6 Conclusions

We studied the problem of cooperatively performing a set of t-tasks with dependencies in a decentralized setting where the communication medium is subject to dynamic changes. We pursued competitive analysis and presented a tight upper bound on the competitive ratio of our randomized algorithm Modified-RS on k-level task t-DAG. When the tasks are independent our results subsume the results of [5] and this bound is tight for the case of independent tasks. We show that the performance of any scheduling algorithm for leveled task graphs depends the computational width that captures the dynamics of the communication medium and on the nature of dependencies among the tasks. In particular we show that performance of any algorithm in this model can deteriorate as the size of the set of independent tasks reduces.

References

[1] B. Chlebus, R. De Prisco, and A.A. Shvartsman. Performing tasks on restartable message-passing processors. *Distributed Computing*, 14(1):49–64, 2001.

[2] R. De Prisco, A. Mayer, and M. Yung. Time-optimal message-efficient work performance in the presence of faults. In *Proceedings of the* 13th *ACM Symposium on Principles of Distributed Computing (PODC 1994)*, pages 161–172, 1994.

[3] S. Dolev, R. Segala, and A.A. Shvartsman. Dynamic load balancing with group communication. In *Proceedings of the* 6th *International Colloquium on Structural Information and Communication Complexity (SIROCCO 1999)*, pages 111–125, 1999.

[4] C. Dwork, J. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *SIAM Journal on Computing*, 27(5):1457–1491, 1998. A preliminary version appears as “Accomplishing work in the presence of failures” in the *Proceedings of the* 11th *ACM Symposium on Principles of Distributed Computing (PODC 1992)*, pages 91–102, 1992.

[2] If the condition $\forall s_i^j, \forall v : s_i^j < v < s_i^{j+1}, \sum_{s_i^j < v < s_i^{j+1}} h(v) < \frac{\alpha_j t}{c_j}$ at level l_j then the competitive ratio will be no worse than $1 + cw\left(1 - \sum_{i=1}^{i=j} \alpha_i\right)$.

[5] Ch. Georgiou, A. Russell, and A.A. Shvartsman. Work-competitive scheduling for cooperative computing with dynamic groups. In *SIAM Journal on Computing: Volume* 34*, Issue 4*, pages 848–862, 2005.

[6] Ch. Georgiou, A. Russell, and A.A. Shvartsman. Work-competitive scheduling for cooperative computing with dynamic groups. In *Proceedings of the* 35th *Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 251–258, 2003.

[7] G. Malewicz, A. Russell, and A. A. Shvartsman. Distributed cooperation during the absence of communication. In *Proceedings of the* 14th *International Symposium on Distributed Computing (DISC 2000)*, pages 119–133, 2000.

[8] D. Powell, editor. *Special Issue on Group Communication Services*, volume 39(4) of *Communications of the ACM.* ACM Press, 1996.

[9] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[10] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.