Cent. Eur. J. Comp. Sci. • 1(1) • 2011 • 19-35
DOI: 10.2478/s13537-011-0002-4

Central European Journal of **Computer Science**

On clustering DAGs for task-hungry computing platforms*

Research Article

Gennaro Cordasco[1]†, Arnold L. Rosenberg[2]‡, Mark Sims[3]§

1 *Dip. di Informatica ed Applicazioni, Università degli Studi di Salerno, Fisciano, 84084, Italy*

2 *Electrical & Computer Engineering, Colorado State University, Fort Collins, CO 80523, USA*

3 *Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003, USA*

Received 23 Nov 2010; accepted 27 Feb 2011

Abstract: Many modern computing platforms are "task-hungry": their performance is enhanced by always having as many tasks available for execution as possible. *IC-scheduling*, a master-worker framework for executing static computations that have intertask dependencies (modeled as DAGS), was developed with precisely the goal of rendering a computation-DAG's tasks eligible for execution at the maximum possible rate. The current paper addresses the problem of enhancing IC-scheduling so that it can accommodate the varying computational resources of different workers, by clustering a computation-DAG's tasks, while still producing eligible (now, clustered) tasks at the maximum possible rate. The task-clustering strategies presented exploit the structure of the computation being performed, ranging from a strategy that works for any DAG, to ones that build increasingly on the explicit structure of the DAG being scheduled.

Keywords: scheduling DAGS • scheduling for task-hungry computing platforms • scheduling for heterogeneous computing platforms

1. Introduction

Many modern computing platforms are "task-hungry": their performance is enhanced by always having as many tasks available for execution as possible. Among the "hungry" modern platforms are systems for the several modalities of Internet-based, or, Cloud computing (*IC*, for short)—including Grid computing (cf. [2, 9]), global and desktop computing

* *A portion of this research was presented at the 37th Intl. Conf. on Parallel Processing (ICPP), 2008.*
† *E-mail: cordasco@dia.unisa.it (Corresponding author)*
‡ *E-mail: rsnbrg@cs.umass.edu*
§ *E-mail: msims@cs.umass.edu*

(cf. [3, 12]), and volunteer computing (cf. [13])—and aggressively multi-core and proposed exascale architectures. Ongoing work [4, 6, 15–17] has been developing *IC-scheduling*, a master-worker framework for executing static computations that have intertask dependencies (modeled as DAGS) in a way that enhances the rate at which a computation-DAG's tasks are rendered eligible for execution. IC-scheduling is motivated by the intuition that rendering tasks eligible as fast as possible—a purely DAG-theoretic goal—will enhance the performance of a "task-hungry" platform, no matter what the (instantaneous) distribution of its constituent computer's speeds. (These speeds can change dynamically, especially when workers are not dedicated to working on the current DAG.) This motivating intuition has been tested in two simulation studies—[14], which studies four large real scientific computations, and [11], which studies hundreds of artificially generated computations of varied sizes. Both studies superimpose a computational model on the purely DAG-oriented framework of IC-scheduling (so that one can talk about makespan), and they compare the makespans of simulated executions of DAGS: each tested DAG is executed by IC-scheduling and by a variety of common scheduling heuristics, including the FIFO strategy used by the Condor system[1]. Both studies suggest that IC-scheduling can accelerate a broad range of computations under a broad range of worker-arrival patterns, by $5\% - 30\%$, or more.
IC-scheduling has evolved from the case studies of [16, 17] to the algorithmic framework of [4, 6, 15], and it can now schedule *IC-optimally*—i.e., *in a manner that produces eligible tasks optimally quickly*—a large repertoire of significant "real" computations; cf. [5]. There remain, though, practically important topics that the framework has yet to address; one such is the topic of this paper. The computers in most modern "task-hungry" platforms often differ significantly in computing power/resources. In volunteer computing, for instance, one might encounter workers who are running x86-based pc's and others who are running multi-core, multi-GHz ones; in the same vein, in Grid computing, the resources allocated to a remotely originating computation may be whatever is not needed for critical local work; heterogeneity is likewise almost ubiquitous in large multi-core systems. Yet, the version of IC-scheduling in the cited sources envisions a scenario in which the IC master allocates one task at a time to each worker computer as it becomes available. Following the lead of systems-oriented studies such as [12], we have begun to study how to enable IC-scheduling to make a computation's tasks *multi-granular* in order to accommodate workers' computational heterogeneity. We specify the idealized computation using fine-grain tasks, and we *cluster* tasks as necessary in order to allocate larger chunks of work to more capable workers. (Our use of task-clustering expands that method's traditional use to avoid communication, as described in sources such as [10].) Our first venture in this direction provided *ad hoc* task-clustering strategies for the computations studied in [5]. The focus of this paper is a variety of *systematic* task-clustering strategies that apply to broad families of computations that admit IC-optimal schedules. We strive for techniques that allow us to choose the size of a *clustered task*—hence, its coarseness—*dynamically,* to acknowledge that we often do not know *a priori* what resources a worker will have at each of its arrivals. Restricting attention to computations that admit IC-optimal schedules limits significantly the class of computations that our strategies can deal with (many computations do not admit any IC-optimal schedule [15])—but this class has been shown to include a broad range of (especially) scientific real computations, including, e.g., all of those discussed in [5]. In compensation, we provide nonobvious, computationally beneficial clustering strategies for computations that do admit IC-optimal schedules.

1.1. Main results

We develop a number of task-clustering strategies, acknowledging that some may be more desirable than others in certain environments. (Multiple strategies are desirable because, e.g., minimizing *inter-worker communication* may be more important with certain computations than others.) Under all of our strategies, *the residual computation after every task-clustering always admits an IC-optimal schedule*—so that we retain the algorithmic benefits of the framework, even as we accommodate workers' heterogeneity. Our first strategy (Section 3.2.1.A) creates a clustered task of any desired size, by exploiting just the order of task-executions of (any) one of the computation's IC-optimal schedules. All of our other strategies exploit aspects of the detailed structure of the computation, as exposed by the suite of algorithms developed in [15] for crafting IC-optimal schedules. These strategies range from ones that work on any computation that admits an IC-optimal schedule (Sections 3.2.1B, 3.2.2) to ones that depend on the very detailed structure of the computation (Sections 3.3.2, 3.3.3).

[1] *Condor Project, University of Wisconsin. http://www.cs.wisc.edu/condor*

2. An overview of IC-scheduling

2.1. Basic concepts

2.1.1. Computation-DAGs

A *(computation-)*DAG $\mathcal{G}$ has a set $\mathcal{N}_\mathcal{G}$ of *nodes*, each representing a *task*, and a set $\mathcal{A}_\mathcal{G}$ of *arcs*, each representing an intertask dependency. For arc $(u \rightarrow v) \in \mathcal{A}_\mathcal{G}$: (a) task v cannot be executed until task u is; (b) u is a *parent* of v, and v is a *child* of u in $\mathcal{G}$. The *indegree* (resp., *outdegree*) of $u \in \mathcal{N}_\mathcal{G}$ is its number of parents (resp., children). A parentless task is a *source*; a childless task is a *sink*. Arcs $(u_1 \rightarrow v_1), (u_2 \rightarrow v_2) \in \mathcal{A}_\mathcal{G}$ are *independent* if they share neither source nor sink. DAG $\mathcal{G}$ is *bipartite* if $\mathcal{N}_\mathcal{G}$ can be partitioned into X and Y so that each arc $(u \rightarrow v)$ has $u \in X$ and $v \in Y$. $\mathcal{G}$ is *connected* if it is so when one ignores arc orientations. When $\mathcal{N}_{\mathcal{G}_1} \cap \mathcal{N}_{\mathcal{G}_2} = \emptyset$, the *sum* $\mathcal{G}_1 + \mathcal{G}_2$ of DAGs $\mathcal{G}_1$ and $\mathcal{G}_2$ is the DAG with node-set $\mathcal{N}_{\mathcal{G}_1} \cup \mathcal{N}_{\mathcal{G}_2}$ and arc-set $\mathcal{A}_{\mathcal{G}_1} \cup \mathcal{A}_{\mathcal{G}_2}$.

2.1.2. Schedules and their quality

When one executes a DAG $\mathcal{G}$, a task $v \in \mathcal{N}_\mathcal{G}$ becomes ELIGIBLE (for execution) only after all of its parents have been executed. (Hence, sources are always ELIGIBLE.) We do not allow recomputation of tasks, so a task loses its ELIGIBLE status once it is executed. In compensation, executing task $v \in \mathcal{N}_\mathcal{G}$ may render new tasks ELIGIBLE; this occurs when v is their last parent to be executed. A *schedule* for $\mathcal{G}$ is a rule for selecting which ELIGIBLE task to execute at each step of an execution of $\mathcal{G}$. (One task is executed at each *step*.) We measure the quality of an execution of $\mathcal{G}$ by the number of ELIGIBLE tasks after each task-execution—the more, the better. (Note: *time is measured in an event-driven manner*, as the number of tasks that have been executed to that point.) Our goal is to execute $\mathcal{G}$'s tasks in an order that maximizes the rate of producing ELIGIBLE tasks *at every step of the execution. A schedule that achieves this demanding goal is* **IC-optimal**.

2.2. IC-Optimal Schedules via DAG Structure

We review the relevant basics of IC-scheduling. Note first that we lose no generality by seeking an IC-optimal schedule for a DAG $\mathcal{G}$ among schedules that execute all of $\mathcal{G}$'s nonsinks before any of its sinks [15].

2.2.1. Scheduling by DAG-decomposition

The priority relation $\triangleright$

For $i = 1, 2$, let DAG $\mathcal{G}_i$ have s_i nonsinks, and let it admit the IC-optimal schedule Σ_i. For each integer t, $E_{\Sigma_i}(t)$ denotes the number of ELIGIBLE nonsources on $\mathcal{G}_i$ at step t. Say that the inequalities in the following system hold[2]:

$$(\forall x \in [0, s_1])\ (\forall y \in [0, s_2])\ \Big[E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \ \leq\ E_{\Sigma_1}(\min\{s_1, x+y\}) + E_{\Sigma_2}(\max\{0, x+y-s_1\})\Big]. \qquad (1)$$

Then $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \triangleright \mathcal{G}_2$. Informally, one never decreases the rate of producing ELIGIBLE tasks by executing a source of $\mathcal{G}_1$ whenever possible. *The relation $\triangleright$ is transitive; one can decide in time $O(s_1 s_2)$ whether $\mathcal{G}_1 \triangleright \mathcal{G}_2$* [15].

Building complex DAGs via composition

We inductively define DAG *composition*.

- Start with a set $\mathcal{B}$ of base DAGs (*BDs*, for short).
  The BDs considered in [4, 15] are *connected bipartite* DAGs.
- Compose $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{B}$ by merging some k sources of $\mathcal{G}_2$ with some k sinks of $\mathcal{G}_1$.
  We denote this operation by $\Uparrow$ and say that $\mathcal{G}$ is *composite of type* $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$.

[2] *$[a, b]$ denotes the set of integers $\{a, a+1, \ldots, b\}$.*

- Add the DAG $\mathcal{G}$ thus obtained to the base set $\mathcal{B}$.

One shows easily that *the composition operation is associative* [15][3].

Scheduling using composition and $\triangleright$-priority

The DAG $\mathcal{G}$ is a $\triangleright$-*linear composition* of the BDs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ if: (a) $\mathcal{G}$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$; ($b$) $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$, for all $i \in [1, n-1]$. Thus, the ordering imposed on $\mathcal{G}$'s BDs by $\triangleright$-priority is consistent with the partial order imposed by topological dependences (i.e, $\mathcal{G}$'s arcs).

Theorem 2.1 ([15]).

Let $\mathcal{G}$ be a $\triangleright$-linear composition of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, where each $\mathcal{G}_i$ admits an IC-optimal schedule Σ_i. The schedule for $\mathcal{G}$ that proceeds as follows is IC optimal.

1. *For $i = 1, \ldots, n$, in turn, execute the tasks of $\mathcal{G}$ that correspond to nonsinks of $\mathcal{G}_i$ in the order mandated by Σ_i.*
2. *Finally, execute all sinks of $\mathcal{G}$ in any order.*

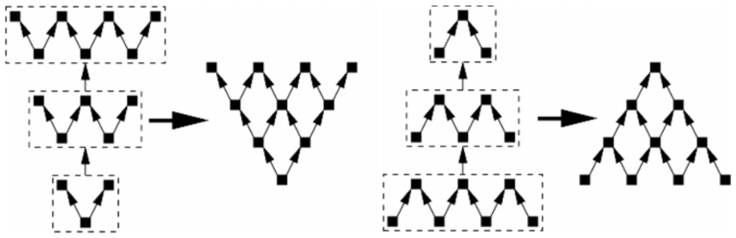

Figure 1. The out-mesh and in-mesh as compositions of BDs.

A suite of algorithms in [15] determine whether a given DAG $\mathcal{G}$ can be decomposed into a set of BDs $\{\mathcal{G}_i \mid i \in [1, n]\}$ that satisfy Theorem 2.1. In brief, whenever possible, the algorithms decompose $\mathcal{G}$ into BDs, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, and describe the dependences among the $\mathcal{G}_i$ via a *super*-DAG $\mathcal{S}_\mathcal{G}$ whose tasks are the BDs and whose arcs form a blueprint of the compositions that created $\mathcal{G}$; cf. Figs. 1, 2. The final algorithm determines whether there is a $\triangleright$-linearization of the BDs that is consistent with the topological dependences within the super-DAG $\mathcal{S}_\mathcal{G}$. Two simple examples will illustrate why super-DAGs are important in analyzing composite DAGs. Fig. 1 depicts two common genres of mesh-DAGs: *out-meshes,* whose arcs point away from the origin task (the unique source) and *in-meshes,* whose arcs point toward the origin task (the unique sink). Both genres give rise to super-DAGs that are paths (or, one-dimensional meshes). Fig. 2 depicts the

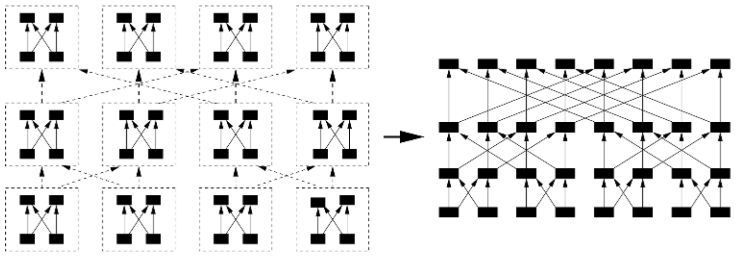

Figure 2. The FFT DAG as a composition of "butterfly" BDs.

[3] *Thus, $\mathcal{G}$ is composite of type $[[\mathcal{G}_1 \Uparrow \mathcal{G}_2] \Uparrow \mathcal{G}_3]$ if, and only if, it is composite of type $[\mathcal{G}_1 \Uparrow [\mathcal{G}_2 \Uparrow \mathcal{G}_3]]$.*

FFT DAG, which encapsulates the data dependencies of the Fast-Fourier Transform algorithm [8]. The super-DAG in this case is a smaller version of the FFT DAG.

Some useful BDs and their inter-priorities

A repertoire of useful BDs will afford us several interesting instantiations of Theorem 3.2. The family of composite DAGS produced by the following BDs (from [15]) is shown in [5, 15] to encompass a wide variety of significant computations ranging from the *Fast-Fourier* and *Discrete-Laplace Transforms* to *matrix-multiplication*, and computational paradigms ranging from *wavefront* computations to computations employing the *divide-and-conquer* and *parallel-prefix* (or, *scan*) operators (cf. [1, 8]).

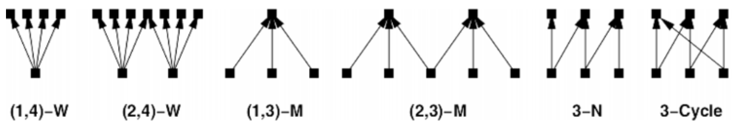

Figure 3. A useful repertoire of (bipartite) BDs.

We proceed left to right along Fig. 3. The first three DAGS are named for the Latin letters suggested by their topologies. W-DAGS epitomize "expansive," and M-DAGS epitomize "reductive" computations.

W-DAGS

For each integer $d > 1$, the $(1, d)$-*W*-DAG $\mathcal{W}_{1,d}$ has one source and d sinks; its d arcs connect the source to each sink. Inductively, for integers $a, b > 0$, the $(a + b, d)$-W-DAG $\mathcal{W}_{a+b,d}$ is obtained from the (a, d)-W-DAG $\mathcal{W}_{a,d}$ and the (b, d)-W-DAG $\mathcal{W}_{b,d}$ by identifying (or, merging) the rightmost sink of the former DAG with the leftmost sink of the latter.

M-DAGS

For integers $a, d > 0$, the (a, d)-M-DAG is obtained by reversing all arcs of the (a, d)-W-DAG.

N-DAGS

For each integer $s > 0$, the s-*N*-DAG $\mathcal{N}_s$ has s sources and s sinks; its $2s - 1$ arcs connect each source v to sink v and to sink $v + 1$ if the latter exists.

(Bipartite) Cycle-DAGS

For each integer $s > 1$, the s-*(Bipartite) Cycle*-DAG $\mathcal{C}_s$ is obtained from $\mathcal{N}_s$ by adding a new arc from the rightmost source to the leftmost sink—so that each source v has arcs to sinks v and $v + 1 \bmod s$.
A broad range of "real" computations can be described as compositions of the preceding BDs; e.g., we see from: (*a*) Fig. 1 that (*i*) the k-sink out-mesh is built by composing, from the source outward, $\mathcal{W}_{1,2}$, then $\mathcal{W}_{2,2}$, ..., then $\mathcal{W}_{k,2}$; (*ii*) the k-source in-mesh is built from its sources upward using $\mathcal{M}_{k,2}$, then $\mathcal{M}_{k-1,2}$, ..., then $\mathcal{M}_{1,2}$; (*b*) Fig. 2 that the FFT DAG is built from its sources outward by composing copies of $\mathcal{C}_2$; (*c*) Fig. 4 that the expansion (resp., reduction) portion

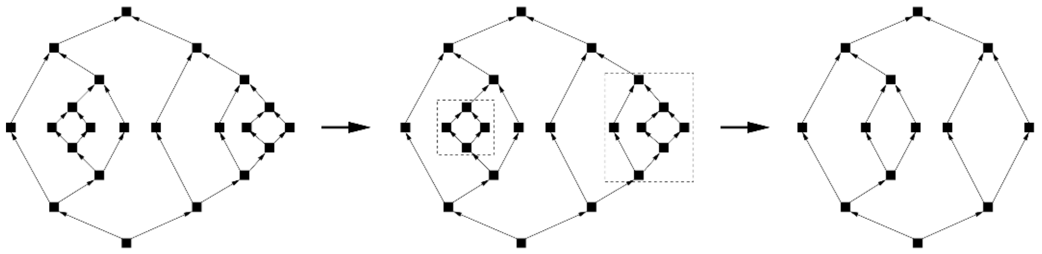

Figure 4. Clustering tasks in a sample expansion-reduction computation.

of an expansion-reduction-DAG is constructed by composing copies of $\mathcal{W}_{1,2}$ (resp., $\mathcal{M}_{1,2}$). The BD-specific results in the sequel build on the following pairwise $\triangleright$-priorities among our BDs.

Theorem 2.2 ([15]).
We observe the following $\triangleright$-priorities among the BDs just defined.

1. *For all integers $s, d > 0$, $\mathcal{W}_{s,d} \triangleright \mathcal{G}$ for the following BDs $\mathcal{G}$: W-DAGS $\mathcal{W}_{s',d'}$ when (a) $[d'<d]$, (b) $\big[[d'=d]$ and $[s' \geqslant s]\big]$; all M-DAGS, N-DAGS, Cycle-DAGS.*
2. *For all integers $s > 0$, $\mathcal{N}_s \triangleright \mathcal{G}$ for the following BDs $\mathcal{G}$: all N-DAGS and M-DAGS.*
3. *For all integers $s > 0$, $\mathcal{C}_s \triangleright \mathcal{G}$ for the following BDs $\mathcal{G}$: $\mathcal{C}_s$; all M-DAGS.*
4. *For all integers $s, d > 0$, $\mathcal{M}_{s,d} \triangleright \mathcal{M}_{s',d'}$ when (a) $[d'>d]$, (b) $\big[[d'=d]$ and $[s' \leqslant s]\big]$.*

Many of our results—especially those that exploit the detailed structure of the DAG $\mathcal{G}$ being scheduled—require that $\mathcal{G}$ be a $\triangleright$-linear composition of BDs, and, moreover, that we are presented with $\mathcal{G}$ in a way that exposes the structure that Theorem 2.1 uses to craft an IC-optimal schedule. One thereby arrives at a *standard* presentation of $\mathcal{G}$ that simultaneously provides an IC-optimal schedule Σ for $\mathcal{G}$:

$$\begin{array}{l} \mathcal{G} \;\text{ is composite of type }\; \mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n \\ \text{where: } \left\{ \begin{array}{l} \mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_n \\ \text{each }\; \mathcal{G}_i \text{ admits an IC-optimal schedule } \Sigma_i. \end{array} \right. \end{array} \tag{2}$$

2.2.2. Duality-based scheduling tools

The *dual* of DAG $\mathcal{G}$ is the DAG $\widetilde{\mathcal{G}}$ obtained by reversing all of $\mathcal{G}$'s arcs. One can infer both IC-optimal schedules and $\triangleright$-priorities for $\mathcal{G}$ from corresponding entities for $\widetilde{\mathcal{G}}$. A *dual* schedule to a schedule Σ for $\mathcal{G}$, is a schedule for $\widetilde{\mathcal{G}}$ that executes, in the reverse order, the sequence of nonsources of $\widetilde{\mathcal{G}}$ that become ELIGIBLE when Σ executes $\mathcal{G}$.

Theorem 2.3 ([4]).
(a) Let the DAG $\mathcal{G}$ admit the IC-optimal schedule Σ. Every schedule that is dual to Σ is IC optimal for $\widetilde{\mathcal{G}}$.
(b) For all DAGS $\mathcal{G}_1$ and $\mathcal{G}_2$: $\mathcal{G}_1 \triangleright \mathcal{G}_2$ if and only if $\widetilde{\mathcal{G}}_2 \triangleright \widetilde{\mathcal{G}}_1$.

3. Accommodating heterogeneity by clustering tasks

3.1. A formal approach to task clustering

Focus on a DAG $\mathcal{G}$ with an associated IC-optimal schedule Σ. Our goal is to determine how to cluster $\mathcal{G}$'s tasks so that their granularities accommodate workers' various computing resources. This goal is inspired by systems-oriented sources (cf. [10, 12]) that strive for kindred goals in an *ad hoc* manner (but, of course, for arbitrary DAGS). Specifically, when a worker P becomes available, we (the IC master) wish to allocate P a *clustered task,* i.e., a set $F \subseteq \mathcal{N}_{\mathcal{G}}$, whose size (relative to other allocated [clustered] tasks) is appropriate for P's computing power. (For instance, as in [12], we might want each worker to take the same time to compute its task.) Having done this, we are left with the *residual* DAG $\mathcal{G}^{[F]}$, i.e., the induced subDAG of $\mathcal{G}$ on the task-set $\mathcal{N}_{\mathcal{G}} \setminus F$.
We restrict attention to clustered tasks F that are *self-contained*, in the sense that the worker that receives F can complete it with no further communications. Thus, every task in F must either be an ELIGIBLE task or a non-ELIGIBLE task together with all of its ancestors back to a set of ELIGIBLE tasks. Fig. 5 illustrates a clustered task F within an out-mesh $\mathcal{G}$ and the resulting residual DAG $\mathcal{G}^{[F]}$.
In detail, we seek strategies for producing clustered tasks F that enjoy the following characteristics.

1. Characteristics that we always require—which all of our strategies achieve:
   (a) F should (almost) match each worker's resources. This means that we should be able to find clustered tasks of (almost) every size.
   (b) F's residual DAG $\mathcal{G}^{[F]}$ should admit an IC-optimal schedule.
2. Characteristics that are desirable, but often not achievable simultaneously:
   (a) F should render ELIGIBLE at least as many tasks of $\mathcal{G}$ as would any other $|F|$-task clustered task.
   (b) The communications that excise F from $\mathcal{G}$ and transmit the results from executing F to the workers executing $\mathcal{G}^{[F]}$ should be as small as possible. (Communication cost is measured by the number of arcs that must be "cut" to separate F from $\mathcal{G}^{[F]}$.)

   Other desiderata may exist in certain settings.

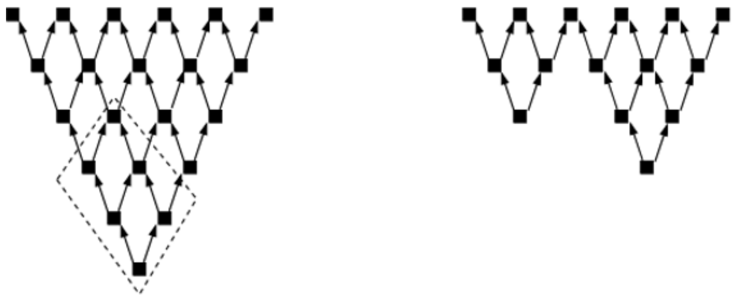

Figure 5. (Left) An out-mesh $\mathcal{G}$ with a clustered task F within the dotted rectangle. (Right) The residual DAG $\mathcal{G}^{[F]}$.

3.2. General task-clustering strategies

3.2.1. The direct *task-clustering strategy*

Our first strategy for clustering tasks applies to any DAG that admits an IC-optimal schedule. The strategies that we develop subsequent to Section 3.2.1.A demand that the IC-optimal schedules emerge from the algorithmic framework of [15], specifically, from Theorem 2.1.

A. A general version of direct task-clustering

Because a schedule Σ for DAG $\mathcal{G}$ associates each task of $\mathcal{G}$ with its order of execution, Σ can be viewed as a one-to-one map of $\mathcal{N}_{\mathcal{G}}$ onto the set $[1, |\mathcal{N}_{\mathcal{G}}|]$. We begin our study with the *direct task-clustering strategy,* which assembles tasks into a clustered task in order of their execution by Σ. Thus, the k-task *direct-clustered task* F_k is the set $F_k = \{\Sigma^{-1}(1), \ldots, \Sigma^{-1}(k)\}$. One can clearly construct clustered tasks of any desired size in this way.

Theorem 3.1.
Let $\mathcal{G}$ be a DAG *that admits the IC-optimal schedule Σ, and let F_k be any direct-clustered task for $\mathcal{G}$. (a) F_k is* self-contained*. (b) We can find a k-task direct-clustered task F_k for any desired k. (c) F_k is* productive*; i.e., it renders maximally many tasks* ELIGIBLE*. (d) The residual* DAG *$\mathcal{G}^{[F_k]}$ admits an IC-optimal schedule. Specifically, the length-$(|\mathcal{N}_{\mathcal{G}}| - k)$ "tail" of Σ is an IC-optimal schedule for $\mathcal{G}^{[F_k]}$.*

The qualifier "any" reflects the fact that $\mathcal{G}$ may admit many IC-optimal schedules, each leading to a different k-task *direct-clustered task* F_k.

Proof Sketch. Parts (a, b, c) follow by definition. Part (d) yields to a cut-and-paste argument. If $\mathcal{G}^{[F_k]}$ did not admit an IC-optimal schedule, then, in particular, the schedule Σ' for $\mathcal{G}^{[F_k]}$ that is the length-$(|\mathcal{N}_{\mathcal{G}}| - k)$ tail of Σ (viewed as a linearization of $\mathcal{G}$'s tasks) would not be IC optimal for $\mathcal{G}^{[F_k]}$. (After executing the tasks in F_k, Σ' mimics Σ.) Replacing Σ' within Σ by a schedule for $\mathcal{G}^{[F_k]}$ that is better at some step would create a schedule that is better for $\mathcal{G}$ than the IC-optimal schedule Σ. This is impossible by definition of IC optimality! □

B. Direct task-clustering for $\triangleright$-linear composite DAGs

We focus henceforth on DAGs $\mathcal{G}$ whose structure is given by (2), and on the IC-optimal schedule for $\mathcal{G}$ produced via Theorem 2.1. For these DAGs, the direct task-clustering strategy can be discerned in $\mathcal{G}$'s structure, as detailed in the following corollary to Theorem 3.1.

Corollary 3.1.
Let $\mathcal{G}$ and Σ satisfy (2). The following procedure creates a productive clustered task F, together with a residual DAG *$\mathcal{G}^{[F]}$ that admits an IC-optimal schedule.*
Add all sources of $\mathcal{G}_1$ in the order mandated by Σ_1, then all sources of $\mathcal{G}_2$ in the order mandated by Σ_2, and so on, until the desired clustered-task size is reached or until every $\mathcal{G}_i$'s sources have been added. In the former case, we are through; in the latter case, add sinks of $\mathcal{G}$ in any order until either no sinks remain or a clustered task of the desired size is achieved.

$\mathcal{G}^{[F]}$ will generally not be a $\triangleright$-linear composition of BDs: removing a clustered task can destroy the required relationship between topological order and $\triangleright$-priority order. Our quest is for techniques that construct clustered tasks that preserve the relationship, so that $\mathcal{G}^{[F]}$ *is* a $\triangleright$-linear composition.

3.2.2. A staggered strategy for $\triangleright$-linear composite DAGs

Our quest for context-specific task-clustering strategies is inspired by the direct task-clustering strategy of Section 3.2.1.B. We assume henceforth that DAG $\mathcal{G}$ satisfies (2) and that its IC-optimal schedule comes via Theorem 2.1. The clustered tasks that we create now satisfy all of our required criteria—*except, perhaps, productivity*.

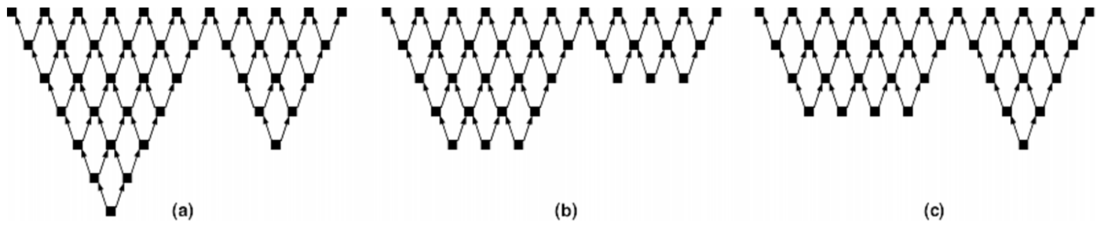

Figure 6. Illustrating why non-direct-clustered tasks might be desirable: (a) a DAG $\mathcal{G}$; (b, c) residual DAGs after excising 6-task clustered tasks.

A word about productivity will motivate the upcoming material. The direct task-clustering strategy satisfies all of our explicit goals for a strategy, including productivity. That said, additional quality criteria warrant exploring other avenues toward clustering tasks. One prime such criterion is to control the amount of *communication* needed to allocate a clustered task and to receive the results of executing that task. Let us concretize our discussion by considering the DAG $\mathcal{G}$ in Fig. 6a. Easily, $\mathcal{G}$ is composite of type

$$\mathcal{W}_{1,2} \Uparrow \mathcal{W}_{1,2} \Uparrow \mathcal{W}_{2,2} \Uparrow \mathcal{W}_{2,2} \Uparrow \mathcal{W}_{3,2} \Uparrow \mathcal{W}_{3,2} \Uparrow \mathcal{W}_{4,2} \Uparrow \mathcal{W}_{4,2} \Uparrow \mathcal{W}_{5,2} \Uparrow \mathcal{W}_{6,2} \Uparrow \mathcal{W}_{10,2},$$

hence, by Theorem 2.2, it is a $\triangleright$-linear composition, as witnessed by the following chain:

$$\mathcal{W}_{1,2} \triangleright \mathcal{W}_{1,2} \triangleright \mathcal{W}_{2,2} \triangleright \mathcal{W}_{2,2} \triangleright \mathcal{W}_{3,2} \triangleright \mathcal{W}_{3,2} \triangleright \mathcal{W}_{4,2} \triangleright \mathcal{W}_{4,2} \triangleright \mathcal{W}_{5,2} \triangleright \mathcal{W}_{6,2} \triangleright \mathcal{W}_{10,2}. \quad (3)$$

Theorem 2.1 turns the chain (3) into an IC-optimal schedule Σ for $\mathcal{G}$ that "bounces back and forth" between $\mathcal{G}$'s left and right out-mesh subDAGs. Specifically, Σ executes:

1. the two copies of $\mathcal{W}_{1,2}$ in some order,
2. the two copies of $\mathcal{W}_{2,2}$ in some order,
3. the two copies of $\mathcal{W}_{4,2}$ in some order,
4. the copy of $\mathcal{W}_{5,2}$, followed by the copy of $\mathcal{W}_{6,2}$, followed by the copy of $\mathcal{W}_{10,2}$.

Now, let us remove a 6-task clustered task F from $\mathcal{G}$. The clustered task $F^{(\mathrm{dir})}$ produced by the direct strategy comprises the two copies of $\mathcal{W}_{1,2}$ and the two copies of $\mathcal{W}_{2,2}$ from the head of chain (3). The resulting residual DAG $\mathcal{G}^{[F^{(\mathrm{dir})}]}$, which appears in Fig. 6b, is a $\triangleright$-linear composition; to wit:

$$\mathcal{W}_{3,2} \triangleright \mathcal{W}_{3,2} \triangleright \mathcal{W}_{4,2} \triangleright \mathcal{W}_{4,2} \triangleright \mathcal{W}_{5,2} \triangleright \mathcal{W}_{6,2} \triangleright \mathcal{W}_{10,2},$$

hence, $\mathcal{G}^{[F^{(\mathrm{dir})}]}$ admits an IC-optimal schedule. However, one needs to "cut" eight arcs of $\mathcal{G}$ in order to excise $F^{(\mathrm{dir})}$ from $\mathcal{G}$, representing the eight data that must be communicated from the worker that executes $F^{(\mathrm{dir})}$ to the master after the task is completed. As an alternative to $F^{(\mathrm{dir})}$, consider the clustered task F consisting of the 6-task prefix of the lefthand out-mesh of $\mathcal{G}$—which contains one copy each of $\mathcal{W}_{1,2}$, $\mathcal{W}_{2,2}$, and $\mathcal{W}_{3,2}$. The resulting residual DAG, $\mathcal{G}^{[F]}$, which appears in Fig. 6c, is also a $\triangleright$-linearization; to wit:

$$\mathcal{W}_{1,2} \triangleright \mathcal{W}_{2,2} \triangleright \mathcal{W}_{3,2} \triangleright \mathcal{W}_{4,2} \triangleright \mathcal{W}_{4,2} \triangleright \mathcal{W}_{5,2} \triangleright \mathcal{W}_{6,2} \triangleright \mathcal{W}_{10,2},$$

hence, $\mathcal{G}^{[F]}$ too admits an IC-optimal schedule. Moreover, one needs "cut" only *six* arcs of $\mathcal{G}$ in order to excise F from $\mathcal{G}$. Thus, producing/executing F requires less communication than producing/executing $F^{(\mathrm{dir})}$! This example motivates the work we describe next.

Note

Viewed as DAGs rather than sets, both $F^{(\mathrm{dir})}$ and F are $\triangleright$-linear compositions:

$$\left[F^{(\mathrm{dir})} : \mathcal{W}_{1,2} \triangleright \mathcal{W}_{1,2} \triangleright \mathcal{W}_{2,2} \triangleright \mathcal{W}_{2,2}\right] \quad \text{and} \quad \left[F : \mathcal{W}_{1,2} \triangleright \mathcal{W}_{2,2} \triangleright \mathcal{W}_{3,2}\right].$$

Hence, they too admit IC-optimal schedules. Of course, this observation is significant only if the worker that receives the clustered task may parcel it out to other workers.

The preceding example suggests that the specifics of the context in which one is clustering tasks influence the relative desirability of available task-clustering strategies. If, e.g., productivity is deemed essential, then one would not consider the alternative strategy just exemplified. But, if minimizing communication is a major concern, then one might well abjure productivity and employ a strategy such as the alternative one.

The preceding alternative task-clustering strategy is not an *ad hoc* one that was designed specifically for $\mathcal{G}$. In fact the strategy is a *staggered* variant of the direct strategy, that can "skip over" constituent BDs as it composes a clustered task. Indeed, not only does the transitivity of $\triangleright$-priority guarantee that the residual DAG will always admit an IC-optimal schedule, it ensures also that the clustered task itself (viewed as a DAG) will admit an IC-optimal schedule. The *staggered* task-clustering strategy composes a clustered task F by always incorporating *entire* BDs from $\mathcal{G}$ into F, in the following manner.

Lemma 3.1.

Let the DAG $\mathcal{G}$ satisfy (2). The following procedure creates a clustered task F such that both the induced subDAG of $\mathcal{G}$ on task-set F and the residual DAG $\mathcal{G}^{[F]}$ admit IC-optimal schedules.

Add all sources of some $\mathcal{G}_{i_1}$ in the order mandated by Σ_{i_1}, then all sources of some $\mathcal{G}_{i_2}$ where $i_2 > i_1$ in the order mandated by Σ_{i_2}, and so on, ending with some sources of some $\mathcal{G}_{i_k}$.

Proof. When the clustered task F is removed from $\mathcal{G}$, the induced DAG on task-set F is composite of type $\mathcal{G}_{i_1} \Uparrow \cdots \Uparrow \mathcal{G}_{i_\ell}$ for some $\ell \geqslant 1$. The residual DAG $\mathcal{G}^{[F]}$ is then composite of type $\mathcal{G}_{j_1} \Uparrow \cdots \Uparrow \mathcal{G}_{j_m}$, where $j_1 < \cdots < j_m$, and $\{j_1, \ldots, j_m\} = [1, n] \setminus \{i_1, \ldots, i_\ell\}$. Invoking (2), the fact that $\mathcal{G}_1 \triangleright \cdots \triangleright \mathcal{G}_n$, and the transitivity of $\triangleright$-priority shows that $[\mathcal{G}_{j_1} \triangleright \cdots \triangleright \mathcal{G}_{j_m}]$ and $[\mathcal{G}_{i_1} \triangleright \cdots \triangleright \mathcal{G}_{i_\ell}]$. Theorem 2.1 completes the proof. □

3.3. Families that are *universal* donors

Our final task-clustering strategy exploits $\mathcal{G}$'s detailed structure, including its specific constituent BDs. We identify families of DAGs that are *universal donors* (of clustered tasks), in the sense that removing *any* clustered task from any of these DAGs leaves a residual DAG that admits an IC-optimal schedule.

3.3.1. The enabling framework

We focus on a DAG $\mathcal{G}$ that satisfies (2) and its IC-optimal schedule Σ that results from Theorem 2.1. We seek to remove a clustered task F from $\mathcal{G}$ by removing a clustered task $F_i \subseteq F$ from each $\mathcal{G}_i$ in $\mathcal{G}$'s $\triangleright$-chain in such a way that $\mathcal{G}^{[F]}$ admits an IC-optimal schedule. We begin by deriving a useful sufficient condition for $\mathcal{G}^{[F]}$ to admit an IC-optimal schedule.

Theorem 3.2.
Let $\mathcal{G}$ be composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$. Let $F = F_1 \cup \cdots \cup F_k$ be a clustered task for $\mathcal{G}$, where, for each $i \in [1,k]$, $F_i \subseteq \mathcal{N}_{\mathcal{G}_i}$. The residual DAG $\mathcal{G}^{[F]}$ admits an IC-optimal schedule whenever the following hold, for some $k \leqslant n$.
Each residual DAG $\mathcal{G}_i^{[F_i]}$ is a sum $\mathcal{G}_i^{[F_i]} = (\mathcal{G}_{i,1} + \cdots + \mathcal{G}_{i,n_i}) + (\{v_1\} \cup \cdots \cup \{v_{m_i}\})$, of BDs (the $\mathcal{G}_{i,j}$) and isolated tasks (the v_ℓ), where, for all $i \in [1,k]$ and $a \in [1,n_i]$:

1. *$\mathcal{G}_{i,a} \triangleright \mathcal{G}_i$;*
2. *$\mathcal{G}_{q,r} \triangleright \mathcal{G}_{i,a}$ for each parent $\mathcal{G}_{q,r}$ of $\mathcal{G}_{i,a}$ in the super-DAG of $\mathcal{G}_i^{[F_i]}$, i.e., for each DAG $\mathcal{G}_{i,a}$ that composes directly with $\mathcal{G}_{q,r}$ as $\mathcal{G}_i^{[F_i]}$ is being formed;*
3. *for all $b \in [1,n_i]$: every pair of BDs, $\mathcal{G}_{i,a}$ and $\mathcal{G}_{i,b}$, are* $\triangleright$-comparable: *either $[\mathcal{G}_{i,a} \triangleright \mathcal{G}_{i,b}]$ or $[\mathcal{G}_{i,b} \triangleright \mathcal{G}_{i,a}]$.*

Proof. Because F contains tasks only from BDs $\mathcal{G}_1, \ldots, \mathcal{G}_k$, the residual DAG $\mathcal{G}^{[F]}$ contains a subDAG that is composite of type $\mathcal{G}_{k+1} \Uparrow \cdots \Uparrow \mathcal{G}_n$. (Recall that, by assumption, $\mathcal{G}_{k+1} \triangleright \cdots \triangleright \mathcal{G}_n$.)

1. For every $j \in [1,k]$ and every residual summand $\mathcal{G}_{j,h}$ of $\mathcal{G}_j^{(F_j)}$, we have $\mathcal{G}_{j,h} \triangleright \mathcal{G}_{j+1}^{(F_{j+1})}$. This follows from the transitivity of $\triangleright$, because $\mathcal{G}_{j,h} \triangleright \mathcal{G}_j$ and $\mathcal{G}_j \triangleright \mathcal{G}_{j+1}$.
2. By hypothesis, $[\mathcal{G}_{q,r} \triangleright \mathcal{G}_{i,j}]$ for any $\mathcal{G}_{q,r}$ with $q \leqslant k$ and $r \leqslant n_q$ such that $\mathcal{G}_{i,j}$ composes directly with $\mathcal{G}_{q,r}$.
3. Also by hypothesis, we have $\triangleright$-comparability among all sibling summands in each sum $(\mathcal{G}_{i,1} + \cdots + \mathcal{G}_{i,n_i})$.

Putting these three facts together, we find that

- the prefix of $\mathcal{G}^{[F]}$, call it $\mathcal{G}'$, that resulted from $\mathcal{G}_1, \ldots, \mathcal{G}_k$ is a $\triangleright$-linear composition;
- the suffix of $\mathcal{G}^{[F]}$, call it $\mathcal{G}''$, that involves only $\mathcal{G}_{k+1}, \ldots, \mathcal{G}_n$ is a $\triangleright$-linear composition;
- $\mathcal{G}' \triangleright \mathcal{G}''$.

Thus, $\mathcal{G}^{[F]}$ is a $\triangleright$-linear composition, so it admits an IC-optimal schedule. □

3.3.2. Reticulated DAGs and universal donorship

We are now ready to expose certain classes $\mathbf{C}$ of DAGs as *universal donors (of clustered tasks)*: for every DAG $\mathcal{G} \in \mathbf{C}$ and every clustered task F for $\mathcal{G}$, the residual DAG $\mathcal{G}^{[F]}$ is a $\triangleright$-linear composition of BDs that admit IC-optimal schedules. We focus only on the four families of BDsdiscussed earlier: W-, N-, Cycle-, and M-DAGs. It is shown in [5, 16, 17] that these BDs compose to form a large variety of computationally significant DAGs. The reader can readily adapt our analyses to many other classes of BDs and the DAGs they produce.

The persistent *priority relation* $\vec{\triangleright}$

A DAG $\mathcal{G}_1$ has *persistent* priority over a DAG $\mathcal{G}_2$, denoted $\mathcal{G}_1 \vec{\triangleright} \mathcal{G}_2$, if the following holds.

- $\mathcal{G}_1 \triangleright \mathcal{G}_2$;
- for every two (self-contained) clustered tasks, $F_1 \subseteq \mathcal{N}_{\mathcal{G}_1}$ and $F_2 \subseteq \mathcal{N}_{\mathcal{G}_2}$, we have $\mathcal{G}_1^{(F_1)} \triangleright \mathcal{G}_2^{(F_2)}$.

One can strengthen some assertions in Theorem 2.2 to discuss *persistent* priority; details are left to the reader.

Lemma 3.2.

We observe the following $\vec{\triangleright}$-priorities among W-, N-, Cycle-, and M-DAG BDs.

- $\mathcal{W}_{s,d} \vec{\triangleright} \mathcal{W}_{s',d'}$ *when* $d > d'$;
- $\mathcal{W}_{s,d} \vec{\triangleright} \mathcal{N}_{s'}$ *when* $d > 2$;
- $\mathcal{W}_{s,d} \vec{\triangleright} \mathcal{M}_{s',d'}$ *when* $d > 2$;
- $\mathcal{W}_{s,d} \vec{\triangleright} \mathcal{C}_{s'}$ *when* $d > 2$.

Reticulated DAG*s*

The following property is sufficient to make several families of composite DAGs universal donors. We say that a DAG $\mathcal{G}$ that satisfies (2) and is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$ is *reticulated* if there is a drawing of $\mathcal{G}$ (called a *standard* drawing if it exists) such that:
If $\mathcal{G}_i$ composes directly with $\mathcal{G}_j$—so that $\mathcal{G}_i$ is a child of $\mathcal{G}_j$ in $\mathcal{G}$'s super-DAG—then: *either* $\mathcal{G}_j \vec{\triangleright} \mathcal{G}_i$ *or* there is a set of *non-crossing, independent* arcs that maps $\mathcal{G}_j$'s sources to a set of sources of $\mathcal{G}_i$ that are consecutive in the drawing. In detail, say that $\mathcal{G}_j$'s sources appear in the drawing in the left-to-right order s_{j_1}, ..., $s_{j_{k_j}}$ and that $\mathcal{G}_i$'s sources appear in the left-to-right order s_{i_1}, ..., $s_{i_{k_i}}$. Then there exists an index $\ell > 0$ such that $\mathcal{A}_{\mathcal{G}}$ contains ($k_j = |\mathcal{S}_{\mathcal{G}_j}|$) arcs, $(s_{j_1}, s_{i_\ell}), \ldots, (s_{j_{k_j}}, s_{i_{\ell+k_j-1}})$, where s_{i_ℓ}, ..., $s_{i_{\ell+k_j-1}}$ are sources of $\mathcal{G}_i$ that appear left to right, in that order. We allow two sources of $\mathcal{G}_j$ to share two consecutive children among $\mathcal{G}_i$'s sources; we do *not* allow an arc from a source of $\mathcal{G}_j$ to a source of $\mathcal{G}_i$ that is not in the set $\{s_{i_\ell}, ..., s_{i_{\ell+k_j}}\}$.

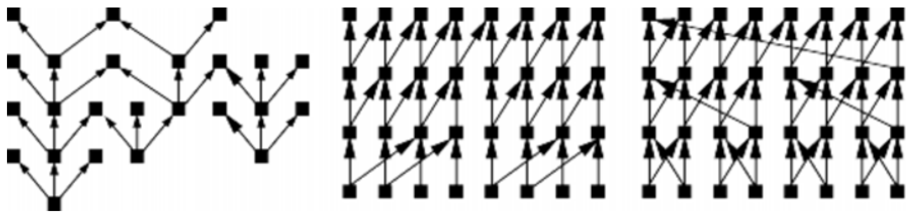

Figure 7. Reticulated compositions of (left) W-DAGs, (middle) N-DAGs, (right) Cycle-DAGs.

When a clustered task connects to a source from a child-BD in a reticulated DAG, it also connects to at least one source from one of that child's parent-BDs. Fig. 7 illustrates sample reticulated DAGs that are compositions of W-, N-, and Cycle-DAGs. Note that for the DAG in Fig. 7(left), we do not need any mapping between the sources of BDs $\mathcal{W}_{3,3}$ and $\mathcal{W}_{2,2}$, because $\mathcal{W}_{3,3} \vec{\triangleright} \mathcal{W}_{2,2}$.
With the BDs we consider now, the residual DAG $\mathcal{G}^{[F]}$ is a sum of other BDs, possibly plus some isolated tasks. In every case, the isolated tasks arise when F removes some of $\mathcal{G}$'s sources without their children (which are sinks of $\mathcal{G}$). For brevity, we do not mention these isolated tasks henceforth, other than to acknowledge their existence.
Call a clustered task F for a BD $\mathcal{B}$ *compact* if all of the sources of $\mathcal{B}$ that F contains are consecutive in the standard drawing of $\mathcal{B}$ depicted in Fig. 3; i.e., the set of sources has the form $\{k, k+1, \ldots, k+\ell\}$ for integers $k \geqslant 0$ and $\ell \geqslant 0$. For Cycle-DAGs, we understand "consecutive" in a cyclic sense, so that the set of sources has the form $\{k, k+1 \bmod m, \ldots, k+\ell \bmod m\}$ for integers $k \geqslant 0$, $\ell > 0$, and $m > 0$. Of course, any clustered task for $\mathcal{B}$ is a sum of compact ones.

A. Compositions of W-dags or of Cycle-dags or of N-dags

Call the subject dags *pure*. We expose the effects of removing a clustered task from each of the subject BDs.

Lemma 3.3.

Removing a clustered task F from an outdegree-d W-dag $\mathcal{W}$ leaves a residual dag $\mathcal{W}^{[F]}$ that is a sum of outdegree-d W-dags, possibly plus some isolated tasks.

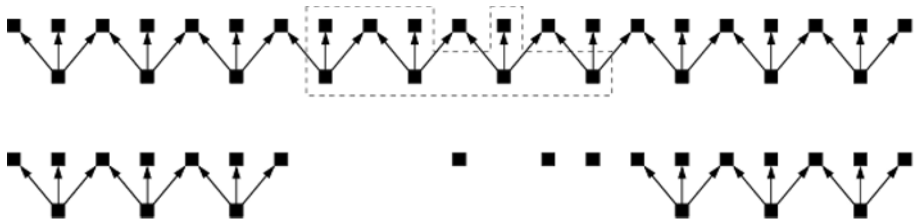

Figure 8. (Top) $\mathcal{W}_{10,3}$, with a clustered task F outlined. (Bottom) $\mathcal{W}^{[F]}_{10,3}$.

Proof Sketch. Focus on a compact clustered task F for $\mathcal{W}$, that possibly contains also some sinks of $\mathcal{W}$; cf. Fig. 8. Excising F leaves one outdegree-d W-dag when F removes one end of $\mathcal{W}$'s sequence of sources, and two disjoint outdegree-d W-dags otherwise. $\mathcal{W}$'s outdegree is preserved in $\mathcal{W}^{[F]}$ because clustered tasks are self-contained. Thus, $\mathcal{W}^{[F]}$ is a sum as indicated in the lemma. By Theorem 2.2: each of $\mathcal{W}^{[F]}$'s summand W-dags has $\triangleright$-priority over $\mathcal{W}$, and each summand is comparable to all others under $\triangleright$-priority. □

Lemma 3.4.

Removing a compact clustered task F from a Cycle-dag $\mathcal{C}$ leaves a residual dag $\mathcal{C}^{[F]}$ that is an outdegree-2 W-dag, possibly plus some isolated tasks.

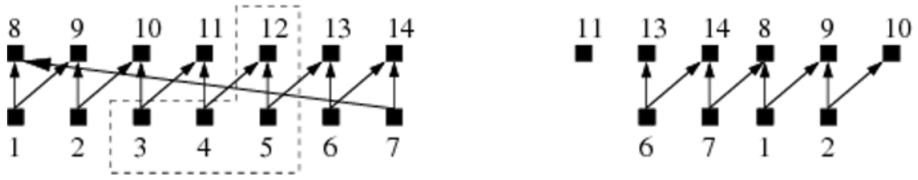

Figure 9. (Left) $\mathcal{C}_7$, with a clustered task F outlined. (Right) $\mathcal{C}^{[F]}_7$, with tasks 1, 2, 8, 9, and 10 shifted to make the residual W-dag more apparent.

Proof Sketch. Fig. 9 illustrates the lemma. The cyclic symmetry of Cycle-dags makes it irrelevant where F resides in the drawing of $\mathcal{C}$. By Theorem 2.2, all W-dags are comparable under $\triangleright$-priority, and every W-dag $\mathcal{W}$ has $\triangleright$-priority over $\mathcal{C}$; i.e., $\mathcal{W} \triangleright \mathcal{C}$. □

Lemma 3.5.

Removing a compact clustered task F from an N-dag $\mathcal{N}$ leaves a residual dag $\mathcal{N}^{[F]}$ that is a sum of zero or more outdegree-2 W-dags, plus zero or one N-dags, possibly plus some isolated tasks.

Proof Sketch. Fig. 10 depicts the generic situation. When F contains the right end (resp., the left end) of $\mathcal{N}$'s sequence of sources, then $\mathcal{N}^{[F]}$ consists of an outdegree-2 W-dag (resp., an N-dag). In general, the residue of $\mathcal{N}$ to the left of F is an outdegree-2 W-dag, and the residue to the right of F is an N-dag. Thus, $\mathcal{N}^{[F]}$ is a sum of the form indicated in the lemma. By Theorem 2.2, $\mathcal{N}^{[F]}$'s summand W-DAG and N-DAG each has $\rhd$-priority over $\mathcal{N}$; moreover, all of the summands are comparable to each other under $\rhd$-priority. $\square$

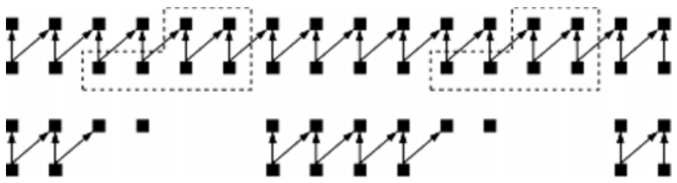

Figure 10. (Top) $\mathcal{N}_{16}$, with a clustered task F outlined. (Bottom) $\mathcal{N}_{16}^{[F]}$.

B. Mixed compositions of W-DAGs, N-DAGs, and Cycle-DAGs

Theorem 3.3.

Every reticulated composition $\mathcal{G}$ of W-, N-, and Cycle-DAGs that satisfies (2) enjoys the following properties.

1. *$\mathcal{G}$ is a universal donor of clustered tasks; let F be one such.*
2. *$\mathcal{G}^{[F]}$ is reticulated and, recursively, is a universal donor of clustered tasks.*

Proof. (a) *$\mathcal{G}^{[F]}$ admits an IC-optimal schedule.* Let $\mathcal{G}$ be as in the theorem. Because: no nondegenerate N-DAG has $\rhd$-priority over any nondegenerate Cycle-DAG and vice versa ("nondegenerate" means "having more than one source") [15], the fact that $\mathcal{G}$ admits an IC-optimal schedule means that its set of constituent BDs—call it $S_{\mathcal{G}}$—either contains no N-DAG or contains no Cycle-DAG. We branch on the preceding alternatives.

Case 1. *$S_{\mathcal{G}}$ contains at least one N-DAG.* Because $S_{\mathcal{G}}$ contains only W- and N-DAGs, Theorem 2.2 tells us that $\mathcal{G}$ must look as follows:

$$\begin{array}{ll} \mathcal{G} \text{ is composite of type} & \mathcal{W}^{(1)} \Uparrow \cdots \Uparrow \mathcal{W}^{(m)} \Uparrow \mathcal{N}^{(1)} \Uparrow \cdots \Uparrow \mathcal{N}^{(n)} \\ \text{where} & \mathcal{W}^{(1)} \rhd \cdots \rhd \mathcal{W}^{(m)} \rhd \mathcal{N}^{(1)} \rhd \cdots \rhd \mathcal{N}^{(n)}. \end{array}$$

Let F be an arbitrary clustered task for $\mathcal{G}$ that contains at least one task from each BD: $\mathcal{W}^{(i_1)}, \ldots, \mathcal{W}^{(i_k)}, \mathcal{N}^{(j_1)}, \ldots, \mathcal{N}^{(j_\ell)}$, where

- $k \geqslant 0$, $\ell \geqslant 0$, and $k + \ell > 0$ (so that F is nonempty);
- $1 \leqslant i_1 < \cdots < i_k \leqslant m$, and $1 \leqslant j_1 < \cdots < j_\ell \leqslant n$.

By Lemmas 3.3, 3.5, then, $\mathcal{G}^{[F]}$ is composite of type $\mathcal{G}^{(1)} \Uparrow \cdots \Uparrow \mathcal{G}^{(k+\ell)}$, where: for $h \in [1, k]$,

$$\mathcal{G}^{(h)} = \begin{cases} \mathcal{W}^{(i_h)} & \text{if } |\mathcal{G}^{(h)}| = |\mathcal{W}^{(i_h)}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes all of } \mathcal{W}^{(i_h)}\text{'s tasks.} & \\ \mathcal{W}^{(h,1)} + \cdots + \mathcal{W}^{(h,a_h)} & \text{if } |\mathcal{G}^{(h)}| < |\mathcal{W}^{(i_h)}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes a strict subset of } \mathcal{W}^{(i_h)}\text{'s tasks and produces a} & \\ \quad \text{sum of some number } a_h > 0 \text{ of W-DAGs.} & \end{cases},$$

while for $h \in [k+1, k+\ell]$,

$$\mathcal{G}^{(h)} = \begin{cases} \mathcal{N}^{(j_{h-k})} & \text{if } |\mathcal{G}^{(h)}| = |\mathcal{N}^{(j_{h-k})}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes all of } \mathcal{N}^{(j_{h-k})}\text{'s tasks.} & \\ \mathcal{W}^{(h,1)} + \cdots + \mathcal{W}^{(h,b_h)} + \mathcal{N}' & \text{if } |\mathcal{G}^{(h)}| < |\mathcal{N}^{(j_{h-k})}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes a strict subset of } \mathcal{N}^{(j_{h-k})}\text{'s tasks and produces a} & \\ \quad \text{sum of some number } b_h > 0 \text{ of W-DAGs, plus, possibly, one N-DAG.} & \end{cases}$$

By invoking Theorem 2.2, we verify the three clauses of Theorem 3.2. We mention only invocations of Theorem 2.2 that refer to our specific situation here.

1. Each BD $\mathcal{N}'$ has lower $\triangleright$-priority than any of its parents in $\mathcal{G}^{[F]}$'s super-DAG, because each such parent is either a W-DAG or another N-DAG.

2. Each parent $\mathcal{B}$ of a generic BD $\mathcal{W}^{(j,a)}$ in $\mathcal{G}^{[F]}$'s super-DAG must be another W-DAG, because $\mathcal{G}$ satisfies (2). Let v be a sink of $\mathcal{B}$, in addition to being a source of $\mathcal{W}^{(j,a)}$ that belongs F. Because F is self-contained, *both* of v's parents (which, recall, are sources of $\mathcal{B}$) belong to F. Consequently, $\mathcal{B}^{[F]}$ is always a sum of W-DAGs, followed on their right by an N-DAG none of whose sinks is part of F. This situation is depicted in Fig. 11. Details are left to the reader.

   Because $\mathcal{G}$ is reticulated, each $\mathcal{W}^{(j,a)}$ has lower $\triangleright$-priority than any of its parents $\mathcal{W}^{(h,r)}$ in $\mathcal{G}^{[F]}$'s super-DAG. To wit, say that $\mathcal{W}^{(j,a)}$ is the residual DAG of some $\mathcal{B}^{(j)}$ and $\mathcal{W}^{(h,r)}$ is the residual DAG of some $\mathcal{B}^{(h)}$. If $\mathcal{B}^{(h)} \vec{\triangleright} \mathcal{B}^{(j)}$, then by definition of relation $\vec{\triangleright}$, we have $\mathcal{W}^{(h,r)} \triangleright \mathcal{W}^{(j,a)}$. Otherwise, because F is self-contained, each source u of $\mathcal{W}^{(q,r)}$ would have an arc (u,v) to some source v of $\mathcal{W}^{(j,a)}$, so that $\mathcal{W}^{(h,r)}$ could have no more sources than $\mathcal{W}^{(j,a)}$ does. Thus, if the relation $\mathcal{B}^{(h)} \triangleright \mathcal{B}^{(j)}$ held before we excised clustered task F, then the relation $\mathcal{W}^{(h,r)} \triangleright \mathcal{W}^{(j,a)}$ will hold after the excision.

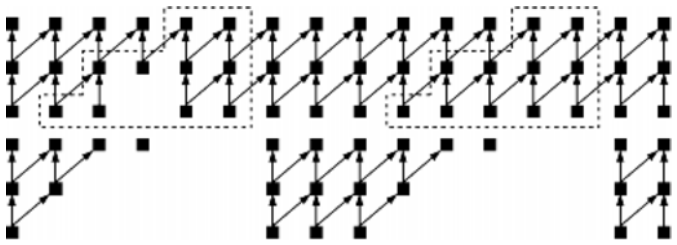

Figure 11. (Top) A reticulated composition of three N-DAGs, of type $\mathcal{N}_3 \Uparrow \mathcal{N}_{13} \Uparrow \mathcal{N}_{16}$. (Bottom) The residual DAG after the outlined two-piece clustered task is removed.

Case 2. *$S_\mathcal{G}$ does not contain a N-DAG.* Because $S_\mathcal{G}$ contains only W- and Cycle-DAGs, Theorem 2.2 tells us that $\mathcal{G}$ must look as follows:

$$\begin{array}{ll} \mathcal{G} \text{ is composite of type} & \mathcal{W}^{(1)} \Uparrow \cdots \Uparrow \mathcal{W}^{(m)} \Uparrow \mathcal{C}^{(1)} \Uparrow \cdots \Uparrow \mathcal{C}^{(n)} \\ \text{where} & \mathcal{W}^{(1)} \triangleright \cdots \triangleright \mathcal{W}^{(m)} \triangleright \mathcal{C}^{(1)} \triangleright \cdots \triangleright \mathcal{C}^{(n)}. \end{array}$$

Let F be an arbitrary clustered task for $\mathcal{G}$ that contains at least one task from each BD $\mathcal{W}^{(i_1)}$, ..., $\mathcal{W}^{(i_k)}$, $\mathcal{C}^{(j_1)}$, ..., $\mathcal{C}^{(j_\ell)}$, where

- $k \geqslant 0$, $\ell \geqslant 0$, and $k + \ell > 0$ (so that F is nonempty).
- $1 \leqslant i_1 < \cdots < i_k \leqslant m$, and $1 \leqslant j_1 < \cdots < j_\ell \leqslant n$.

By Lemmas 3.3, 3.4, then, $\mathcal{G}^{[F]}$ is composite of type $\mathcal{G}^{(1)} \Uparrow \cdots \Uparrow \mathcal{G}^{(k+\ell)}$, where: for $h \in [1,k]$,

$$\mathcal{G}^{(h)} = \begin{cases} \mathcal{W}^{(i_h)} & \text{if } |\mathcal{G}^{(h)}| = |\mathcal{W}^{(i_h)}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes all of } \mathcal{W}^{(i_h)}\text{'s tasks.} & \\ \mathcal{W}^{(h,1)} + \cdots + \mathcal{W}^{(h,a_h)} & \text{if } |\mathcal{G}^{(h)}| < |\mathcal{W}^{(i_h)}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes a strict subset of } \mathcal{W}^{(i_h)}\text{'s tasks and produces} & \\ \quad \text{a sum of some number } a_h > 0 \text{ of W-DAGs} & \end{cases},$$

while for $h \in [k+1, k+\ell]$,

$$\mathcal{G}^{(h)} = \begin{cases} \mathcal{C}^{(j_{h-k})} & \text{if } |\mathcal{G}^{(h)}| = |\mathcal{C}^{(j_{h-k})}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes all of } \mathcal{C}^{(j_{h-k})}\text{'s tasks.} & \\ \mathcal{W}^{(h,1)} + \cdots + \mathcal{W}^{(h,b_h)} & \text{if } |\mathcal{G}^{(h)}| < |\mathcal{C}^{(j_{h-k})}| \\ \quad \text{In this case, } \mathcal{G}^{[F]} \text{ takes a strict subset of } \mathcal{C}^{(j_{h-k})}\text{'s tasks and produces} & \\ \quad \text{a sum of some number } b_h > 0 \text{ of W-DAGs.} & \end{cases}$$

The proof is now completed by mimicking Case 1 and adding the observation that every W-DAG has higher $\triangleright$-priority than any Cycle-DAG. Details are left to the reader.

Note

The proof for a DAG $\mathcal{G}$ that is a *pure* composition of W- or N- or Cycle-DAGs is easily extracted from Cases 1 and 2: pure compositions of W-DAGs correspond to Case 2 when $n = 0$; pure compositions of N-DAGs correspond to Case 1 when $m = 0$; pure compositions of Cycle-DAGs correspond to Case 2 when $m = 0$.

(b) $\mathcal{G}^{[F]}$ *is reticulated.* We claim that when one removes a clustered task F from a reticulated DAG $\mathcal{G}$, the residual DAG $\mathcal{G}^{[F]}$ is reticulated. To see this, focus on BDs $\mathcal{B}^{(j,a)}$ and $\mathcal{B}^{(h,b)}$, where $\mathcal{B}^{(h,b)}$ is a parent of $\mathcal{B}^{(j,a)}$ in $\mathcal{G}^{[F]}$'s super-DAG. Say that $\mathcal{B}^{(j,a)}$ is the residual DAG of some BD $\mathcal{B}^{(j)}$, while $\mathcal{B}^{(h,b)}$ is the residual DAG of some BD $\mathcal{B}^{(h)}$. Because $\mathcal{G}$ is reticulated, there are only two possibilities.

1. If $\mathcal{B}^{(h)} \vec{\triangleright} \mathcal{B}^{(j)}$, then the persistence of $\vec{\triangleright}$-priority guarantees that $\mathcal{B}^{(h,b)} \vec{\triangleright} \mathcal{B}^{(j,a)}$.
2. Otherwise, $\mathcal{A}_{\mathcal{G}}$ must contain all of $\mathcal{B}^{(j)}$'s arcs, so we can focus on the subset $\mathcal{A}_{\mathcal{B}^{(j,a)}} \subseteq \mathcal{A}_{\mathcal{B}^{(j)}}$ that contains all $(u, v) \in \mathcal{A}_{\mathcal{B}^{(j)}}$ such that u is a source of $\mathcal{B}^{(j,a)}$. Because $\mathcal{G}$ is reticulated, there must be a drawing of $\mathcal{G}$ and a set of noncrossing independent arcs that maps $\mathcal{B}^{(j,a)}$'s sources to consecutive sources of $\mathcal{B}^{(h,b)}$.

It follows that $\mathcal{G}^{[F]}$ is reticulated, as claimed. □

3.3.3. Compositions of M-DAGs

M-DAGs are harder to deal with than our other three families of BDs, for reasons suggested by Fig. 12. Compositions

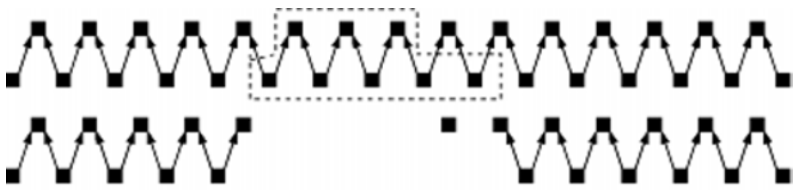

Figure 12. (Top) $\mathcal{M}_{15,2}$, with a clustered task F outlined. (Bottom) $\mathcal{M}^{[F]}_{15,2}$.

of reticulated versions of our other BDs split into sums that are easily described and analyzed when a (not necessarily compact) clustered task is excised:

- Compositions of W-DAGs retain the W-DAG structure.
- Compositions of Cycle-DAGs become compositions of Cycle- and W-DAGs—but the W-DAGs are always ancestors of the Cycle-DAGs in the composition's super-DAG.
- Compositions of N-DAGs become compositions of N- and W-DAGs, but the constituent N- and W-DAGs relate within the super-DAG in a way that is easily described and analyzed.

In contrast, the notion "reticulated" makes no sense with compositions of M-DAGs, and, as suggested by Fig. 12, these compositions can become structurally complicated ensembles of W-, N-, and M-DAGs. Consequently, we approach compositions of M-DAGs via a technique of analysis that depends on the fact that M- and W-DAGs are *dual* to one another (cf. Section 2.2). To simplify exposition, we restrict attention to *indegree*-2 M-DAGs; adapting our result to M-DAGs of arbitrary indegrees is straightforward.
The *closure* $\widehat{F}$ of a clustered task F for $\mathcal{G}$ is the DAG obtained from $\mathcal{G}$ by removing all tasks of $\mathcal{G}^{[F]}$ *except for isolated tasks*. Note that $\widehat{F}$ may differ from F if $\widehat{F}$ contains tasks that are isolated with respect to $\mathcal{G}^{[F]}$ but are connected to some task in F; cf. Fig. 8.

Lemma 3.6.

Let $\mathcal{G}$ be a reticulated composition of W-DAGs that satisfies (2). For every clustered DAG F, the DAG $\widehat{F}$ admits an IC-optimal schedule and is reticulated.

Proof. Note that (2) specialized to compositions of W-DAGS tells us that

$$\begin{array}{ll} \mathcal{G} \text{ is composite of type } & \mathcal{W}^{(1)} \Uparrow \cdots \Uparrow \mathcal{W}^{(n)} \\ \text{where} & \mathcal{W}^{(1)} \rhd \cdots \rhd \mathcal{W}^{(n)}. \end{array}$$

For every clustered DAG F, the DAG $\widehat{F}$ has the form (parentheses added to enhance legibility):

$$\widehat{F} \ = \ \left(\widehat{F}^{(1,1)} + \cdots + \widehat{F}^{(1,i_1)}\right) + \cdots + \left(\widehat{F}^{(k,1)} + \cdots + \widehat{F}^{(k,i_k)}\right),$$

where each $\widehat{F}^{(i,j)}$ is a connected subDAG of $W^{(i)}$. We claim that $\widehat{F}$ is a $\rhd$-linear composition, hence admits an IC-optimal schedule. To see this, note that each $\widehat{F}^{(\ell,h)}$'s $\rhd$-priority does not exceed that of any of its parents in $\mathcal{G}$'s super-DAG; moreover, $\widehat{F}^{(\ell,h)}$ is $\rhd$-comparable to every other $\widehat{F}^{(i,j)}$. We invoke the fact that $\mathcal{G}$ is reticulated and branch on the form of $\widehat{F}^{(\ell,h)}$. If $\widehat{F}^{(\ell,h)}$ is:

- a W-DAG, then we invoke an argument similar to that in the proof of Theorem 3.3;
- an N-DAG, then $\widehat{F}^{(\ell,h)}$'s parents (if any) are W- or N-DAGS, hence of no lower $\rhd$-priority;
- an indegree-2 M-DAG $\mathcal{M}_{s,2}$, then $\widehat{F}^{(\ell,h)}$'s parents (if any) are W- or N-DAGS or indegree-2 M-DAGS $\mathcal{M}_{s',2}$ with $s' > s$.

In all cases, our claims regarding $\rhd$-priority and $\rhd$-comparability follow from Theorem 2.2.
The argument in the proof of Theorem 3.3(2) now verifies that $\widehat{F}$ is reticulated. □

We finally have the desired analogue of Theorem 3.3 for compositions of M-DAGS.

Theorem 3.4.
Let $\mathcal{G}$ be composition of M-DAGS that satisfies (2) and that has a reticulated dual DAG $\widetilde{\mathcal{G}}$. Then $\mathcal{G}$ is a universal donor of clustered tasks.

Proof. Let F be a clustered task for $\mathcal{G}$, and let $\widetilde{F}$ be the clustered task for $\mathcal{G}$'s dual DAG $\widetilde{\mathcal{G}}$ that is obtained by removing all tasks that belong to F's closure $\widehat{F}$. Easily, $\widetilde{\mathcal{G}}^{(\widehat{F})}$ is the induced subDAG of $\widetilde{\mathcal{G}}$ on the task-set $\mathcal{N}_{\widetilde{F}}$; i.e., $\widetilde{\mathcal{G}}^{(\widehat{F})} = \widehat{\widetilde{F}}$. By Lemma 3.6, $\widetilde{\mathcal{G}}^{(\widehat{F})}$ admits an IC-optimal schedule; by Theorem 2.3, then, $\mathcal{G}^{[\widehat{F}]}$ too admits such a schedule. The theorem now follows by noting that $\mathcal{G}^{[\widehat{F}]}$ and $\mathcal{G}^{[F]}$ differ only in that the latter DAG may have some additional isolated tasks—which have no impact on the existence of an IC-optimal schedule. □

4. Conclusions

We have begun to study the problem of clustering the tasks of computation-DAGS so as to accommodate the heterogeneity of worker computers in task-hungry computing environments. Acknowledging the importance of having many clustering strategies that address the specifics of the multitude of such environments, we have developed a range of task-clustering strategies, ranging from those that ignore the specific structure of the computation-DAG being scheduled (Section 3.2.1.A), to those that exploit the gross structure of the DAG (Sections 3.2.1.B and 3.2.2), to those that depend in a very detailed way on the DAG's structure (Sections 3.3.2 and 3.3.3). We hope that this development brings IC-scheduling one step closer to applicability in real computing environments. Important next steps in our research are: (1) to apply the results here to a testbed of significant scientific computation-DAGS, in a variety of computing environments; (2) to determine how to adapt the current research to AREA-oriented scheduling [7], the offshoot of IC-scheduling that applies to *all* computation-DAGS.

Acknowledgments

A. Rosenberg was supported in part by US NSF Grants CCF-0342417, CNS-0615170, and CNS-0905399.

References

[1] Bluestein L.I., A linear filtering approach to the computation of the Discrete Fourier Transform, IEEE TRANS AUDIO ELECTROACOUST, 1970, AU-18, 451-455

[2] Buyya R., Abramson D., Giddy J., A case for economy Grid architecture for service oriented Grid computing, 10th Heterogeneous Computing Wkshp. (23 April 2001 San Francisco USA), IEEE Computer Society, 2001

[3] Cirne W., Marzullo K., The Computational Co-Op: gathering clusters into a metacomputer, 13th Int'l Parallel Processing Symp. (1999 San Juan, Puerto Rico), IEEE Computer Society, 1999, 160-166

[4] Cordasco G., Malewicz G., Rosenberg A.L., Advances in IC-scheduling theory: scheduling expansive and reductive dags and scheduling dags via duality, IEEE T PARALL DISTR, 2007, 18, 1607-1617

[5] Cordasco G., Malewicz G., Rosenberg A.L., Applying IC-scheduling theory to some familiar computations, Wkshp. on Large-Scale, Volatile Desktop Grids (PCGrid'07) (2007, Long Beach, California, USA), IEEE Computer Society, 2007

[6] Cordasco G., Malewicz G., Rosenberg A.L., Extending IC-scheduling via the Sweep algorithm, J PARALLEL DISTR COM, 2010, 70, 201-211

[7] Cordasco G., Rosenberg A.L., On scheduling dags to maximize area, 23rd IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS) (2009, Rome, Italy), IEEE Computer Society, 2009

[8] Cormen T.H., Leiserson C.E., Rivest R.L., Stein C., Introduction to Algorithms, 2nd ed., MIT Press, Cambridge, MA, 2001

[9] Foster I., Kesselman C., The Grid 2: Blueprint for a New Computing Infrastructure, 2nd ed., Morgan-Kaufmann, San Francisco, 2004

[10] Gerasoulis A., Yang, T., A comparison of clustering heuristics for scheduling dags on multiprocessors, J PARALLEL DISTR COM, 1992, 16, 276-291

[11] Hall R., Rosenberg A.L., Venkataramani A., A comparison of dag-scheduling strategies for Internet-based computing, 21st IEEE Int'l Parallel and Distributed Processing Symp.(IPDPS) (2007, Long Beach, California, USA), IEEE Computer Society, 2007

[12] Kondo D., Casanova H., Wing E., Berman F., Models and scheduling mechanisms for global computing applications, Int'l Parallel and Distr. Processing Symp. (IPDPS) (2002, Fort Lauderdale, California), IEEE Computer Society, 2002

[13] Korpela E., Werthimer D., Anderson D., Cobb J., Lebofsky M., SETI@home: massively distributed computing for SETI, In: P.F. Dubois (Ed.), Computing in Science and Engineering, IEEE Computer Soc. Press, Los Alamitos, CA, 2000

[14] Malewicz G., Foster I., Rosenberg A.L., Wilde M., A tool for prioritizing DAGMan jobs and its evaluation, JOURNAL OF GRID COMPUTING, 2007, 5, 197-212

[15] Malewicz G., Rosenberg A.L., Yurkewych M., Toward a theory for scheduling dags in Internet-based computing, IEEE T COMPUT, 2006, 55, 757-768

[16] Rosenberg A.L., On scheduling mesh-structured computations for Internet-based computing, IEEE T COMPUT, 2004, 53, 1176-1186

[17] Rosenberg A.L., Yurkewych M., Guidelines for scheduling some common computation-dags for Internet-based computing, IEEE T COMPUT, 2005, 54, 428-438