A Comparison of Dag-Scheduling Strategies for Internet-Based Computing

Robert Hall Arnold L. Rosenberg Arun Venkataramani
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003, USA

July 5, 2006

Abstract

The temporal unpredictability in both communication and computation encountered when computing over the Internet complicates attempts to efficiently compute collections of jobs that have complex interdependencies. A recently introduced algorithmic scheduling theory aims to mitigate the impact of temporal unpredictability and to counter the unpredictability in the number of remote workers available over time, by sequencing job executions to always keep as large as possible the number of jobs that can be allocated to remote clients. Two simulation experiments are described, which use quite different metrics to evaluate the effectiveness of the theory and its schedules. both experiments indicate that, under a wide range of parameters, schedules produced using the theory significantly improve the execution of a large class of computation-dags, over the schedules produced by three simple, intuitively compelling heuristics.

1 Introduction

Advances in technology have made collections of computers that communicate across the Internet a viable computational platform [7]. Thus, we see many efforts aimed at using multiple distributed computers to solve a single computational problem [1, 2, 3, 12]. Perhaps the major impediment to scheduling complex computations efficiently in this new environment is *temporal unpredictability*.

- Communication takes place over the Internet, hence may experience unpredictable delays.
- Remote computing workers may not be dedicated to performing the work they receive remotely, hence may execute that work at an unpredictable rate.

This uncertainty in timing virtually precludes accurate identification of critical paths in complex computations, and hence demands a new scheduling paradigm that acknowledges the strengths and weaknesses of the Internet as a computational medium.

Several recent papers [17, 18, 15] have identified a new goal when scheduling computations that consist of multiple tasks (henceforth termed *jobs* to emphasize their coarse-grained nature) with complex interdependencies. The third of these sources has begun to develop a scheduling theory for an idealized version of Internet-based computing that attempts to schedule the jobs of a complex computation in a manner that always maximizes the number of jobs that are eligible for assignment to remote workers. One hopes that this goal has the dual advantage of:

- maximally exploiting available remote workers, by minimizing the likelihood that there is no job for a remote worker; and
- minimizing the likelihood of the "gridlock" that can occur when no new jobs can be assigned pending the completion of already assigned jobs.

The scheduling theory—which is currently being extended in [4, 5]—can optimally schedule a large variety of structurally uniform dags such as the four familiar ones depicted in Fig. 1, in addition to a broad repertoire of less uniform ones, such as those depicted in Fig. 2.

The theory being developed in [4, 5, 15] aspires to a scheduling mechanism for complex Internet-based computations that has both a strong theoretical grounding and a signficant impact for scheduling real computations. The current paper continues the effort begun in [13] to evaluate the extent to which the initial phase of the theory, as developed in [15], achieves its two-pronged goal. Our study shares with that in [13] the methodology of a head-to-head competition between schedules that are mandated by the theory of [15] and schedules that arise from intuitively compelling heuristics. Our study differs from that in [13] in four major respects:

1. They employ four dags that arise in "real" scientfic computations as their testbed workload. We employ a large number of artificially generated dags.
2. They use as their benchmark an extension of the optimal scheduling algorithm of [15] that produces schedules for all dags. We use the actual algorithm of [15], which succeeds in scheduling only a certain class of dags; cf. Theorem 2.1. (We ensure that our artificially generated dags belong to this class.)

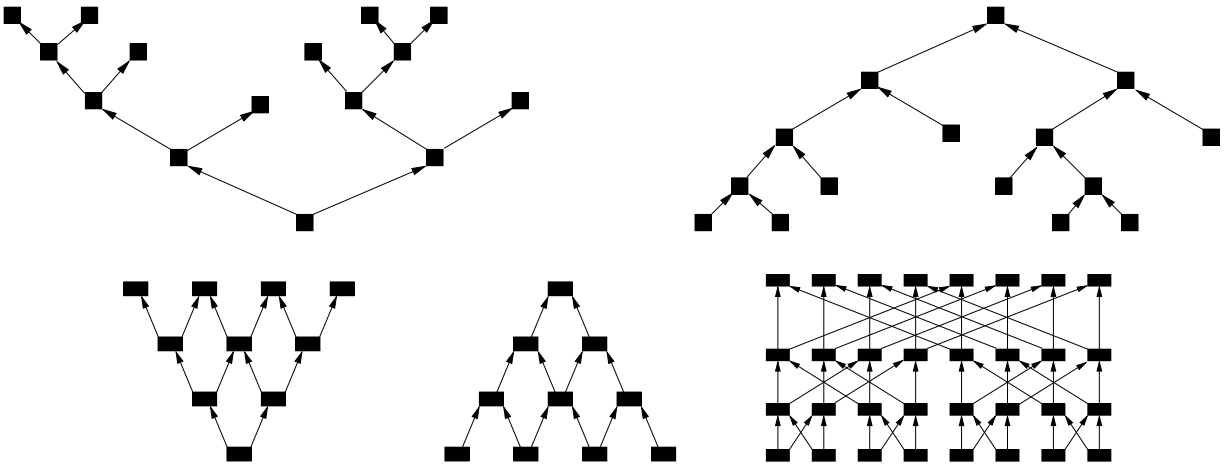

Figure 1: (Top) *An "expansive" and a "reductive" binary tree.* (Bottom) *An "expansive" and a "reductive" 2-dimensional mesh; the FFT dag.*

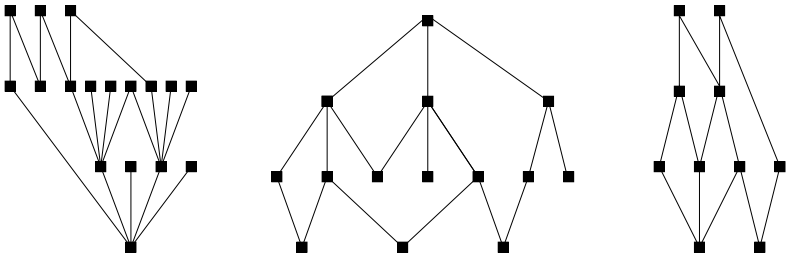

Figure 2: *Three composite dags that the framework of [15] can schedule optimally. Edges represent arcs that point upward.*

3. Their theoretically mandated schedules compete only against a particular FIFO scheduler that is inspired by the regimen used in the Condor system [3]. We have our schedules compete against enhanced version of FIFO, LIFO schedulers, as well as a greedy scheduler that employs the same scheduling criteria locally that our optimal scheduler employs globally.

4. They employ a single metric to compare their competing regimens' schedules, namely, a type of "batched makespan" metric; cf. [14]. We employ both the "batched makespan" metric and an "area-maximizing" metric that captures a notion of the average rate of producing eligible jobs.

Notably, despite these differences, we find similar lessons emerging from [13] and our study—using both quality metrics. For a broad range of situations that one might expect to encounter in Internet-based computing environments, the schedules produced using the theory achieve significant performance improvements over their competitors for a wide

range of system parameters.

Related work. The source that is most closely related to our study is [13], as discussed earlier. In terms of the underlying scheduling theory, the closest source are: [17, 18], wherein the new scheduling paradigm and specify optimal schedules for uniform dags typified by those in Fig. 1 is introduced; [15], wherein the initial version of the theory—whch is what we employ in our experiments—is developed; [14], wherein the *batched* version of the theory, which gives rise to the "batched makespan" quality metric is introduced. Finally, the impetus for our study derives from the exciting systems- and/or application-oriented studies of Internet-based computing, in sources such as [1, 2, 3, 7, 8, 11, 12, 19].

A Roadmap. Section 2 outlines portions of the theory of [15] that are relevant to our study. Section 3 presents the scheduling algorithms whose comparison is our main theme. Section 4 describes the methodology underlying our algorithmic comparisons. Section 5 presents and analyzes our experimental results. Section 6 summarizes our contributions and our future plans regarding this work.

2 A Basis for a Scheduling Theory

2.1 Computation-Dags

A *directed graph* $\mathcal{G}$ is given by a set of *vertices* $N_{\mathcal{G}}$ and a set of *arcs* $A_{\mathcal{G}}$, each of the form $(u \to v)$, where $u, v \in N_{\mathcal{G}}$. A *path* in $\mathcal{G}$ is a sequence of arcs that share adjacent endpoints, as in the following path from vertex u_1 to vertex u_n: $(u_1 \to u_2)$, $(u_2 \to u_3)$, $\ldots$, $(u_{n-2} \to u_{n-1})$, $(u_{n-1} \to u_n)$. A *dag* (*directed acyclic graph*) $\mathcal{G}$ is a directed graph that has no cycles—so that no path of the preceding form has $u_1 = u_n$. When a dag $\mathcal{G}$ is used to model a computation, i.e., is a *computation-dag:*

- each vertex $v \in N_{\mathcal{G}}$ represents a job in the computation;
- an arc $(u \to v) \in A_{\mathcal{G}}$ represents the dependence of job v on job u: v cannot be executed until u is.

(We omit the qualifier "computation-" henceforth.) For any arc $(u \to v) \in A_{\mathcal{G}}$, u is a *parent* of v, and v is a *child* of u in $\mathcal{G}$. The *indegree* (resp., *outdegree*) of vertex u is its number of parents (resp., children). A parentless vertex of $\mathcal{G}$ is a *source*; a childless vertex is a *sink*; all other vertices are *internal*. $\mathcal{G}$ is *connected* if, when arcs' orientations are ignored, there is a path connecting every pair of distinct vertices; $\mathcal{G}$ is *bipartite* if $N_{\mathcal{G}}$ consists entirely of sources and sinks[1]; for reasons that are clarified in Section 2.3, we call a connected bipartite dag a *CBBB*, for *C*onnected *B*ipartite *B*uilding *B*lock.

[1]Perforce, all arcs go from a source to a sink.

2.2 A Model for Executing Dags on the Internet

"Pebble games" on dags have yielded elegant formalizations of a variety of problems related to scheduling dags. Such games use tokens, *pebbles*, to model the progress of a computation on a dag: the placement or removal of the various available types of pebbles—which is constrained by the dependencies modeled by the dag's arcs—represents the changing (computational) status of the dag's job-vertices. Our study is based on the *Internet-Computing* (*IC*, for short) *Pebble Game* of [17]. Based on studies of IC in, e.g., [1, 11, 19], arguments are presented in [17, 18] (*q.v.*) that justify the simplified form of the Game studied here.

A. The rules of the Game. The IC Pebble Game on a dag $\mathcal{G}$ involves one player S, the *Server*, who has access to unlimited supplies of two types of pebbles: ELIGIBLE pebbles, whose presence indicates a job's eligibility for execution, and EXECUTED pebbles, whose presence indicates a job's having been executed. The Game is played as follows.

The IC Pebble Game

- S begins by placing an ELIGIBLE pebble on each unpebbled source of $\mathcal{G}$.
  /*Unexecuted sources are always eligible for execution, having no parents whose prior execution they depend on.*/
- At each step, S
  - selects a vertex that contains an ELIGIBLE pebble,
  - replaces that pebble by an EXECUTED pebble,
  - places an ELIGIBLE pebble on each unpebbled vertex of $\mathcal{G}$ all of whose parents contain EXECUTED pebbles.
- S's goal is to allocate vertices in such a way that every vertex v of $\mathcal{G}$ *eventually* contains an EXECUTED pebble.
  /*This modest goal is necessitated by the possibility that $\mathcal{G}$ is infinite.*/

A *schedule* for the IC Pebble Game is a rule for selecting which ELIGIBLE pebble to execute at each step of a play of the Game. For brevity, we henceforth call a vertex ELIGIBLE (resp., EXECUTED) when it contains an ELIGIBLE (resp., an EXECUTED) pebble. For uniformity, we henceforth talk about executing vertices rather than jobs.

B. The quality of a play of the Game. Our goal is to play the IC Pebble Game in a way that maximizes the production rate of ELIGIBLE vertices. For each step t of a play of the Game on a dag $\mathcal{G}$ under a schedule Σ, we denote by $E_\Sigma(t)$ the number of vertices of $\mathcal{G}$ that are ELIGIBLE at step t.

We measure the **IC quality** *of a play of the IC Pebble Game on a dag $\mathcal{G}$ by the size of $E_\Sigma(t)$ at each step t of the play—the bigger, the better. Our goal is an* **IC-optimal** *schedule Σ, in which $E_\Sigma(t)$ is as big as possible for all steps t.*

The goal of IC quality—hence of IC optimality—stems from the following intuitions. (1) Schedules that produce ELIGIBLE vertices more quickly may reduce the chance of the "gridlock" that could occur when remote clients are slow—so that new jobs cannot be allocated pending the return of already allocated ones. (2) If the IC Server receives a batch of requests for jobs at (roughly) the same time, then having more ELIGIBLE jobs available allows the Server to satisfy more requests, thereby increasing "parallelism."

2.3 A Framework for Crafting IC-Optimal Schedules

The priority relation $\triangleright$. For $i = 1, 2$, let the bipartite dag $\mathcal{G}_i$ have s_i sources, and let it admit the IC-optimal schedule Σ_i. If the following inequalities hold:[2]

$$\begin{array}{l}(\forall x \in [0, s_1])\ (\forall y \in [0, s_2]) : \\ \quad E_{\Sigma_1}(x) + E_{\Sigma_2}(y) \ \leq\ E_{\Sigma_1}(\min\{s_1, x+y\}) + E_{\Sigma_2}(\max\{0, x+y-s_1\}),\end{array} \tag{2.1}$$

then $\mathcal{G}_1$ *has priority over* $\mathcal{G}_2$, denoted $\mathcal{G}_1 \triangleright \mathcal{G}_2$. Informally, one never decreases IC quality by executing a source of $\mathcal{G}_1$ whenever possible.

A framework for scheduling complex dags. The operation of *composition* is defined inductively as follows.

- Start with a set $\mathcal{B}$ of base "building block" dags.[3]
- One composes dags $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{B}$—which could be the same dag with vertices renamed to achieve disjointness—to obtain a composite dag $\mathcal{G}$, as follows.
  - Let $\mathcal{G}$ begin as the *sum* (or, disjoint union), $\mathcal{G}_1 + \mathcal{G}_2$, of the dags $\mathcal{G}_1, \mathcal{G}_2$. Rename vertices to ensure that $N_{\mathcal{G}}$ is disjoint from $N_{\mathcal{G}_1}$ and $N_{\mathcal{G}_2}$.
  - Select some set S_1 of sinks from the copy of $\mathcal{G}_1$ in the sum $\mathcal{G}_1 + \mathcal{G}_2$, and an equal-size set S_2 of sources from the copy of $\mathcal{G}_2$ in the sum.
  - Pairwise identify (i.e., merge) the vertices in the sets S_1 and S_2 in some way. The resulting set of vertices is $\mathcal{G}$'s vertex-set; the induced set of arcs is $\mathcal{G}$'s arc-set.[4]
- Add the dag $\mathcal{G}$ thus obtained to the base set $\mathcal{B}$.

[2] $[a, b]$ denotes the set of integers $\{a, a+1, \ldots, b\}$.
[3] The base dags considered in [4, 15] are CBBBs.
[4] An arc $(u \to v)$ is *induced* if $\{u, v\} \subseteq N_{\mathcal{G}}$.

We denote the composition operation by $\Uparrow$ and say that $\mathcal{G}$ is *composite of type* $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$.

The dag $\mathcal{G}$ is a $\triangleright$-*linear composition* of the CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ if: (a) $\mathcal{G}$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$ (composition is associative); (b) each $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$, for all $i \in [1, n-1]$.

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 Σ for $\mathcal{G}$ that proceeds as follows is IC optimal.*

1. *For $i = 1, \ldots, n$, in turn, Σ executes the vertices of $\mathcal{G}$ that correspond to sources of $\mathcal{G}_i$, in the order mandated by Σ_i.*
2. *Σ finally executes all sinks of $\mathcal{G}$ in any order.*

3 The Four Competing Scheduling Algorithms

3.1 The IC-Optimal Scheduling Algorithm ICO

One finds in [15] a suite of algorithms that determine whether or not a given dag $\mathcal{G}$ can be decomposed into a set of CBBBs $\{\mathcal{G}_i\}$ that satisfy Theorem 2.1 and that, whenever possible, use the theorem to derive an IC-optimal schedule for $\mathcal{G}$. These algorithms, whose assemblage we collectively call scheduler ICO, process $\mathcal{G}$ via the following sequence of steps.

1. "Prune" $\mathcal{G}$ to remove all *shortcut* arcs.
   - An arc $a = (u \to v)$ is a *shortcut* if there is a path from u to v that does not use a.
   - The algorithm in, e.g., [10] will accomplish this "pruning."
   - The resulting "pruned" dag $\mathcal{G}'$ shares its IC-optimal schedules with $\mathcal{G}$.
2. "Parse" $\mathcal{G}'$ into a collection of CBBBs, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, such that $\mathcal{G}'$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$.
   - Such a "parsing" is not always possible. When it is possible, it is unique and can be found by iteratively greedily removing a maximal CBBB subgraph of $\mathcal{G}'$ all of whose sources are sources of $\mathcal{G}'$.
3. Replace $\mathcal{G}'$ by the *super-dag* $\mathcal{G}''$ whose vertices are the CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ and whose arcs form a blueprint of the sequence of compositions that created $\mathcal{G}'$.
   - Specifically, if $\mathcal{G}'$ was formed by identifying some sources of CBBB $\mathcal{G}_i$ with some sinks of CBBB $\mathcal{G}_j$, then there is an arc in $\mathcal{G}''$ from supervertex $\mathcal{G}_j$ to supervertex $\mathcal{G}_j$.
4. Determine whether or not there is an $\triangleright$-linearization of the CBBBs $\mathcal{G}_1, \ldots, \mathcal{G}_n$ that is consistent with the topological dependencies within the super-dag $\mathcal{G}''$.

• This determination basically amounts to determining if the priority relation $\triangleright$ imposes on $\mathcal{G}''$ a total order that is consistent with a topological sort[5] of $\mathcal{G}''$.

5. If all steps to this point have succeeded, then output the schedule for $\mathcal{G}$ mandated by Theorem 2.1.

3.2 The Competing Heuristic Schedulers

The FIFO heuristic.

1. FIFO initially enqueues $\mathcal{G}$'s sources into a FIFO queue Q, in nonincreasing order of outdegree (so that the vertices of maximum outdegree emerge first); vertices of equal outdegree are enqueued in random order.

2. When a remote client C requests a vertex, FIFO allocates to C the vertex obtained by dequeuing Q.

3. When a vertex v completes execution, FIFO enqueues (into Q), in nonincreasing order of outdegree, those of v's children that are rendered ELIGIBLE by v's execution; vertices of equal outdegree are enqueued in random order.

The LIFO heuristic.

1. LIFO initially pushes $\mathcal{G}$'s sources into a (LIFO) stack S, in nondecreasing order of outdegree (so that the vertices of maximum outdegree emerge first); vertices of equal outdegree are pushed in random order.

2. When a remote client C requests a vertex, LIFO allocates to C the vertex obtained by popping S.

3. When a vertex v completes execution, LIFO pushes (onto S), in nondecreasing order of outdegree, those of v's children that are rendered ELIGIBLE by v's execution; vertices of equal outdegree are pushed in random order.

The GREEDY heuristic.

1. GREEDY initially inserts $\mathcal{G}$'s sources, in random order, into a MAX-Priority Queue P [6]. (The ultimate order of vertices having distinct outdegrees is determined by P's queuing discipline.)

[5] A topological sort of a dag is a linearization of its vertices under which all arcs point from left to right.

2. When a remote client C requests a vertex, GREEDY allocates to C the vertex obtained via the EXTRACT-MAX operation on P.

3. When a vertex v completes execution, GREEDY inserts (into P), in random order, those of v's children that are rendered ELIGIBLE by v's execution.

4 The Experimental Setup

This section is devoted to describing the experimental setup that we use to compare ICO against FIFO, LIFO, and GREEDY. Our experiments proceed as follows.

1. We generate a dag $\mathcal{G}$ that is random within a class of dags that admit IC-optimal schedules; Section 4.1 provides details.

2. We execute $\mathcal{G}$ using all four schedulers of Section 3. Since FIFO, LIFO, and GREEDY all involve a degree of randomization, we invoke each fifty times on each dag and use the means and variances of their "performances" for our comparisons with ICO.

3. We compile the statistics that compare the "qualities" of the executions of step 2. We employ two quality metrics for our comparisons, which arise from quite distinct intuitions. The first metric can be viewed as measuring the "average" IC quality of a schedule; the second introduces a computational model and uses an analogue of "time to completion" as its metric. Sections 4.2 and 4.3 provide details.

4.1 On Generating "Random" Dags

Of course, the real test for any scheduler is to deal with given dags of possibly complex structures. However, since our goal is to assess the value of IC optimality when it exists, we "cheat" by evaluating our competing schedulers on dags that are chosen via random compositions from among dags that are guaranteed (by results in [17, 18, 15]) to admit such schedules. Our selection process proceeds as follows.

1. We select a random target size for the dag we want to generate. We select a range of sizes that allows us to observe trends in the behaviors of schedulers, typically from a few hundred vertices to several thousand.

2. We choose an appropriate collection of CBBBs randomly from a repertoire that is defined and analyzed in [15].

   IC-optimal schedules for all these CBBBs are identified in [15], as are all $\triangleright$-priorities among them.

3. We compose the selected CBBBs in ways that are chosen randomly among compositions that are guaranteed (by Theorem 2.1) to preserve IC-optimal schedulability.

   This means that we compose a CBBB $\mathcal{G}_i$ with a CBBB $\mathcal{G}_i$ only when $\mathcal{G}_i \rhd \mathcal{G}_j$.

We now provide the definitions and salient properties of the CBBBs in the repertoire.

4.1.1 The repertoire of CBBBs

Although we choose CBBBs from those in [15], our methodology applies easily to any class of CBBBs that admit IC-optimal schedules, such as those in [4]. Our first three CBBBs are named for the Latin letters suggested by their topologies.

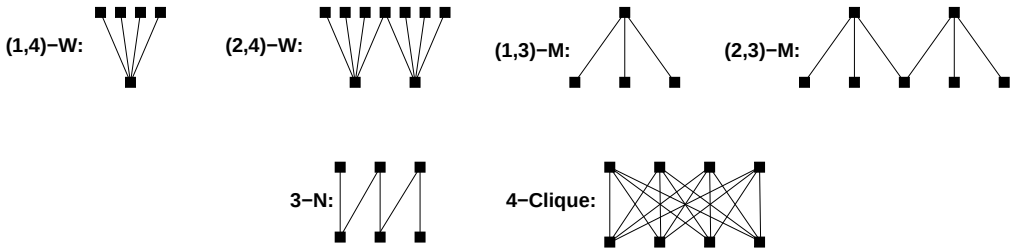

Figure 3: *The building blocks for our semi-uniform dags. Edges represent arcs that point upward.*

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 positive integers a, b, 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 $\mathcal{W}_{a,d}$ with the leftmost sink of $\mathcal{W}_{b,d}$.

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

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. $\mathcal{N}_s$ is obtained from $\mathcal{W}_{s-1,2}$ by adding a new source on the right whose sole arc goes to the rightmost sink. The leftmost source of $\mathcal{N}_s$—the dag's *anchor*—has a child that has no other parents.

(Bipartite) Clique-dags. For each integer $s > 1$, the s-*(Bipartite) Clique-dag* $\mathcal{Q}_s$ has s sources and s sinks, and an arc from each source to each sink.

4.1.2 Schedulability and $\triangleright$-priorities among the CBBBs

The following results from [15] enable our generation of random dags that are assured to admit IC-optimal schedules.

Vertex execution order. We specify an IC-optimal execution order for the sources of our CBBBs. (Sinks can be executed in any way once all sources are EXECUTED [15].)

1. Execute sources of $\mathcal{W}_{a,d}$ and $\mathcal{M}_{a,d}$ from one end to the other.
2. Execute sources of $\mathcal{N}_s$ from the anchor to the other end.
3. Execute sources of Q_s in any order.

$\triangleright$-priorities. One verifies the following pairwise $\triangleright$-priorities among our CBBBs.

1. For all s and d, $\mathcal{W}_{s,d} \triangleright \mathcal{G}$ for the following CBBBs $\mathcal{G}$:
   (a) all W-dags $\mathcal{W}_{s',d'}$ whenever $d' < d$, or whenever $d' = d$ and $s' \geq s$;
   (b) all M-dags and N-dags;
   (c) Clique-dags $\mathcal{Q}_{s'}$ with $s' \leq d$.
2. For all s, $\mathcal{N}_s \triangleright \mathcal{G}$ for the following CBBBs $\mathcal{G}$:
   (a) all N-dags $\mathcal{N}_{s'}$, for all s'; (b) all M-dags.
3. For all s and d, $\mathcal{M}_{s,d} \triangleright \mathcal{M}_{s',d'}$ whenever $d' > d$, or whenever $d' = d$ and $s' \leq s$.
4. For all s, $\mathcal{Q}_s \triangleright \mathcal{Q}_s$.

4.1.3 Generating random compositions of CBBBs

We generated dags for our experiments in the following way.

Selecting random CBBBs. In order to ensure that all dags generated for our experiments admitted IC-optimal schedules, we constructed these dags from random $\triangleright$-linearizable compositions of the CBBBs enumerated in the preceding subsection. In order to ensure a variety of dag structures that abstracted "real" computational dependencies, we employed the following combinations of CBBBs.

1. Random W-dags, abstracting "expansive" dags, that grow from sources to sinks.
2. Random M-dags, abstracting "reductive" dags, that shrink from sources to sinks.
3. Random W-dags, "followed by" N-dags, "followed by" M-dags, abstracting "fork-join" dags, that grow from sources, then shrink toward sinks.
4. Random compositions of $\mathcal{Q}_2$, abstracting convolutional dags such as the FFT dag.

Fig. 1 exemplifies options 1, 2, and 4; Fig. 2 exemplifies option 3.

Randomly composing CBBBs. Having assembled a $\triangleright$-linearizable selection of CBBBs, we composed them in a manner that guaranteed consistency with Theorem 2.1. All selections—of CBBBs, of partially constructed dags to compose, and of sources and sinks to effect compositions—were random in terms of both numbers and selcted individuals.

Executing dags. As noted earlier, the ICO scheduler is deterministic, but the three competing heuristics all incorporate elements of randomness. In recognition of this, we had each of FIFO, LIFO, and GREEDY execute each generated dag fifty times, and we used the means and standard deviations of the results to compare the heuristics' schedules with the IC-optimal one.

4.2 The Area-Maximization Experiment

4.2.1 The area-maximization metric

Our first quality metric for comparing schedulers can be viewed as measuring the "average" IC quality of a schedule. Consider a schedule Σ for an n-vertex dag $\mathcal{G}$.

The notion of IC optimality rewards Σ only if it maximizes the number of ELIGIBLE vertices *at every step* in its execution of $\mathcal{G}$; i.e.,

$$(\forall t) \quad E_\Sigma(t) \;=\; \max_{\Sigma' \text{ a schedule for } \mathcal{G}} \{E_{\Sigma'}(t)\}.$$

The new quality metric we identify here rewards Σ for maximizing the *average* number of vertices of $\mathcal{G}$ that are ELIGIBLE as $\mathcal{G}$ is executed. We term this average the "area" of schedule Σ because of the formal analogy with the integral-calculus technique of approximating integrals—in this case of the function E_Σ—by Riemann sums.

The *plot* of schedule Σ is the $(n+1)$-entry vector $\Pi(\Sigma) \;=\; \langle E_\Sigma(0),\; E_\Sigma(1), \ldots,\; E_\Sigma(n)\rangle$. Of course, $E_\Sigma(0)$ is just the number of sources of $\mathcal{G}$, and $E_\Sigma(n) \equiv 0$ for any schedule, but we retain these entries of $\Pi(\Sigma)$ for completeness. The *area* of schedule Σ is the sum $A(\Sigma) \;=\; \sum_{i=0}^{n} E_\Sigma(i)$. Of course, the normalized area

$$\widehat{E}(\Sigma) \;\stackrel{\text{def}}{=}\; \frac{1}{n} A(\Sigma) \;=\; \frac{1}{n} \sum_{i=0}^{n} E_\Sigma(i). \tag{4.1}$$

is the average number of vertices of $\mathcal{G}$ that are ELIGIBLE under schedule Σ.

We note some important properties of the area-maximization metric.

> **Observation**. **(a)** *Whereas some dags do not admit any IC-optimal schedule [15], every dag admits an area-maximizing schedule.* **(b)** *If the dag $\mathcal{G}$ admits*

an IC-optimal schedule, then: (i) *every area-maximizing schedule for $\mathcal{G}$ is IC optimal;* (ii) *every IC-optimal schedule for $\mathcal{G}$ is area-maximizing.*

4.2.2 The area-maximization experiment

Our area-maximization experiment generates random dags in the manner described in Section 4.1. We study each generated dag $\mathcal{G}$ as follows.

1. We compute $\widehat{E}(\text{ICO})$ directly, as $\mathcal{G}$ is generated.
2. We execute $\mathcal{G}$ fifty times using each of our three heuristic schedulers, and we compile the mean values and standard deviations of $\widehat{E}$ for the fifty schedules produced by each heuristic scheduler. We denote by $\widetilde{E}(\text{FIFO})$, $\widetilde{E}(\text{LIFO})$, and $\widetilde{E}(\text{GREEDY})$, the resulting mean values of $\widehat{E}(\text{FIFO})$, $\widehat{E}(\text{LIFO})$, and $\widehat{E}(\text{GREEDY})$, respectively.

We compare schedulers Σ and Σ' under the area-maximization metric via the quantity

$$\Delta(\Sigma, \Sigma') \stackrel{\text{def}}{=} \widetilde{E}(\Sigma) - \widetilde{E}(\Sigma'),$$

where $\widetilde{E}(\text{ICO}) \equiv \widehat{E}(\text{ICO})$ by convention. (Note that $n \cdot \Delta(\Sigma, \Sigma')$ is just the $l1$ distance between $\Pi(\Sigma)$ and $\Pi(\Sigma')$.) As just observed, $\Delta(\text{ICO}, \Sigma')$ is always nonnegative.

4.3 The Batched-Makespan Experiment

4.3.1 The batched-makespan metric

Our second quality metric for comparing schedulers arises from changing the IC Pebble Game from a "client-centric" to a "server-centric" model of Internet-based computing.

The *"client-centric" model*—which is the one studied in [15, 17, 18]—views the Server as being interrupted by the arrival of each available remote client. When so interrupted, the Server allocates an ELIGIBLE job to the client, if such a job exists; otherwise, the client is viewed as disappearing (looking elsewhere for work).

The *"server-centric" model*—a variant of which is studied in [14]—has remote clients assemble in groups at preassigned times—perhaps, but not necessarily, periodically. At these preassigned times, the Server polls for the presence of a nonempty group of clients and a nonempty pool of ELIGIBLE jobs. When a poll is "successful," finding, say, $r \geq 1$ remote clients and $e \geq 1$ ELIGIBLE jobs, the Server chooses $\max(r, e)$ ELIGIBLE jobs and allocates them, one per client, until either clients or ELIGIBLE jobs run out. At this point, any unserved clients disappear (say, looking elsewhere for work) and any unallocated jobs are returned to the ELIGIBLE pool.

The "server-centric" model suggests the following *batched-makespan metric* for a scheduler Σ when executing an n-vertex dag $\mathcal{G}$. Given a pattern of arrivals of batched requests, $r_0, r_1, \ldots$, with the interpretation that, for each nonnegative integer i, r_i remote clients arrive requesting jobs when the Server polls for the ith time, how many pollings does it take before $\mathcal{G}$ is completely executed? If we assume that pollings are timed so that all jobs that were allocated at the ith polling will have been completed before the $(i+1)$th polling, then *we are seeking the smallest integer m such that $a_0 + a_1 + \cdots + a_m \geq n$, where*

$$\begin{aligned} a_0 &= \max(r_0, E_\Sigma(0)) \\ a_1 &= \max(r_0, E_\Sigma(a_0)) = \max(r_1, E_\Sigma(\max(r_0, E_\Sigma(0)))) \\ &\vdots \quad \vdots \quad \vdots \\ a_m &= \max(r_0, E_\Sigma(a_{m-1})). \end{aligned}$$

Under the "server-centric" model, the Server may have to allocate $r > 1$ ELIGIBLE jobs at once at some preassigned polling times. In contrast, ICO always waits until it sees all $E_{\text{ICO}}(t-1)$ jobs that are ELIGIBLE after the execution of the $(t-1)$th job before selecting the tth job to allocate to a remote client. This leads to the following apparent anomaly.

> **Observation**. *Under the batched-makespan metric, some schedulers can conceivably outperform scheduler* ICO *on some dags.*

4.3.2 The batched-makespan experiment

Our batched-makespan experiment generates random dags in the manner described in Section 4.1. We study the batched-execution of each generated dag $\mathcal{G}$ as follows.

We assume that the Server polls for clients requesting work in an event-driven manner, namely, as soon as all tasks allocated in earlier phases are completed. At each poll by the Server, we assume that there are ρ requests for work, where ρ is a random variable whose value is distributed exponentially in the set $\{2, 4, \ldots, 2^{14}\}$. (Thus, our system has no memory; each polling is independent of all others.) When coupled with the variability in the sizes of the generated dags, this range of values for ρ gives us a reasonable picture of how varying dag sizes and request sizes are dealt with by our four schedulers.

We execute each generated dag $\mathcal{G}$ fifty times using each of our four schedulers. (In contrast to the area-maximization experiment, for this experiment, ICO encounters randomness also, due to the request-arrival rate ρ.) We end up with four *batched execution times*, $T(\text{ICO})$, $T(\text{FIFO})$, $T(\text{LIFO})$, and $T(\text{GREEDY})$, which are the means of the observed numbers of pollings required by each of the four schedulers. For each $\mathcal{G}$, we compare scheduler ICO against its competing schedulers $\Sigma \in \{\text{FIFO}, \text{LIFO}, \text{GREEDY}\}$ via the *phase-ratio* $T(\Sigma) \div T(\text{ICO})$ (so larger reported numbers favor ICO).

5 Experimental Results and Interpretation

5.1 Area-Maximization Results

Our results from the area-maximization experiment present both the means and 95% confidence intervals of the quantities $\Delta(\text{ICO}, \Sigma)$ for $\Sigma \in \{\text{FIFO}, \text{LIFO}, \text{GREEDY}\}$. (In many cases, the intervals are so tight around the means that they are indistinguishable from those points.) We fitted curves of the form $a \cdot v^b$ to the data—where v is the size of the generated dag—using a nonlinear regression function, calculated in GNUplot via the Marquardt-Levenberg algorithm [9].

5.1.1 Schedules for familiar dags

We instantiated the dags of Fig. 1 in several different sizes: 3-to-10 levels for the FFT dag and 10-to-100 levels for the mesh-structured dags (so the largest FFT dag was roughly equal in size to the largest meshes). The results appear in Fig. 4. One discerns a number of meaningful patterns in the figure. Most importantly, $\Delta(\text{ICO}, \Sigma)$ *grows nontrivially with the size of the dag being scheduled.*

FFT dags. GREEDY and FIFO compute the same schedule (to within the random ordering of equal-outdegree vertices), because each nonsink of the FFT has outdegree 2, so the priority-queue of GREEDY functions exactly as the FIFO queue of FIFO. LIFO performs the worst, because whenever both sources of a copy of $\mathcal{C}_2$ within the FFT is completed, LIFO allocates both of the newly ELIGIBLE sinks consecutively, despite the fact that they are sources of distinct copies of $\mathcal{C}_2$. The disparity among the three heuristics manifests itself only via the coefficient of the form av^b. All three yield to an exponent $b = 1.7$; $a_{\text{LIFO}} = 0.11$ in contrast with $a_{\text{FIFO}} = a_{\text{GREEDY}} = 0.023$.

Two-dimensional reduction-meshes. Both GREEDY and FIFO perform well on these dags, but neither attains IC optimality. GREEDY's preference for "inner" mesh-vertices (those of outdegree 2) rather than "outer" ones causes it to interleave the execution of mesh levels; FIFO's random ordering of equal-outdegree vertices precludes its sequential execution of each mesh level. Both of these characteristics preclude IC-optimal schedules [18]. LIFO lags the other schedules, presumably for a reason similar to its behavior on FFT dags.

Two-dimensional evolving meshes. In this case, both GREEDY and FIFO produce IC-Optimal schedules. (Randomizing vertex order seems not to cause suboptimality as long as FIFO's is enforced.) LIFO again lags, for the same reasons as above: most vertices depend directly on more than one parent, and LIFO favors executing nodes that are deeper into the dag, and less likely to have neighboring vertices that are EXECUTED.

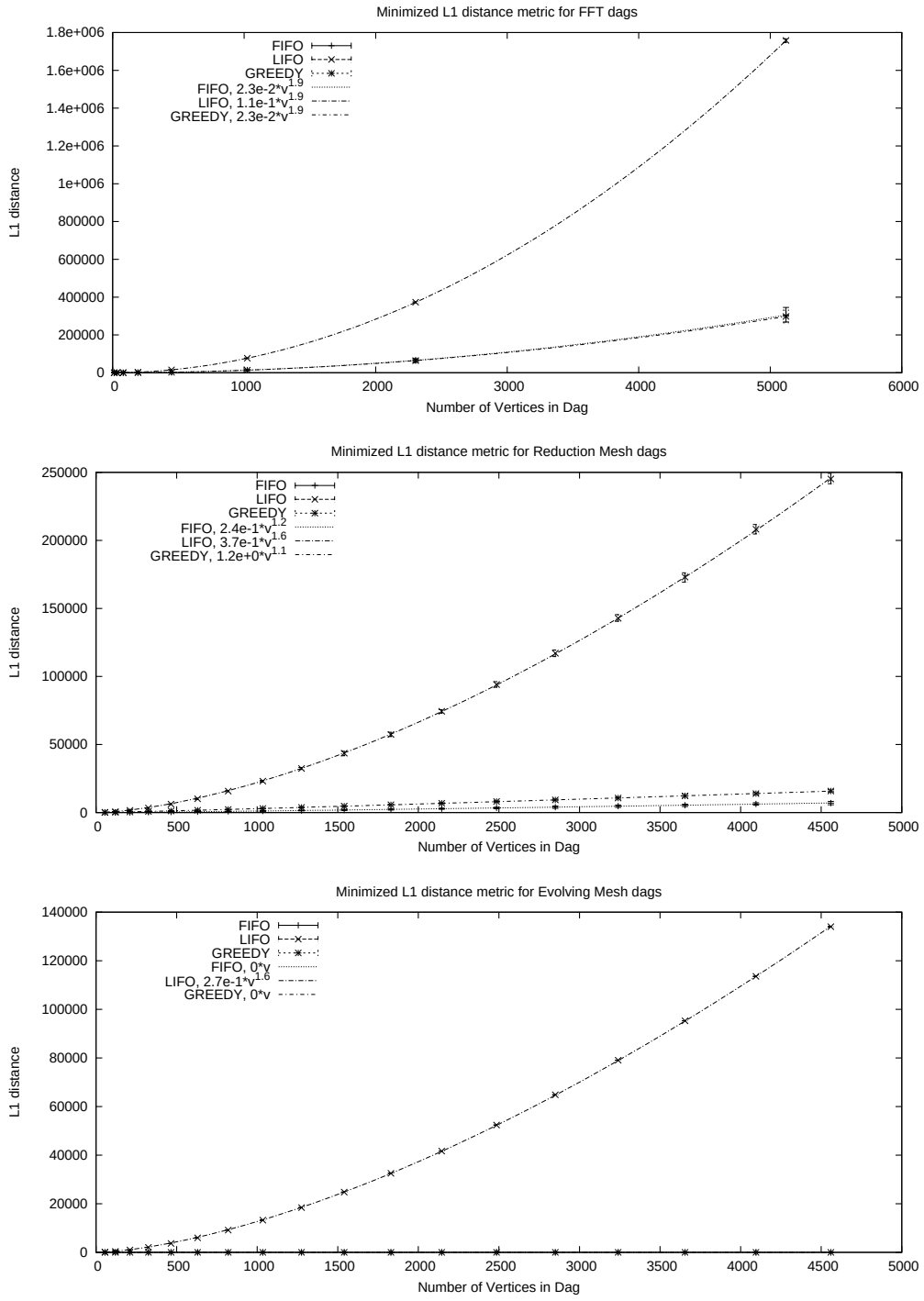

Figure 4: *Area-maximization results for FFT dags (top), reduction-meshes (middle), and evolving meshes (bottom).*

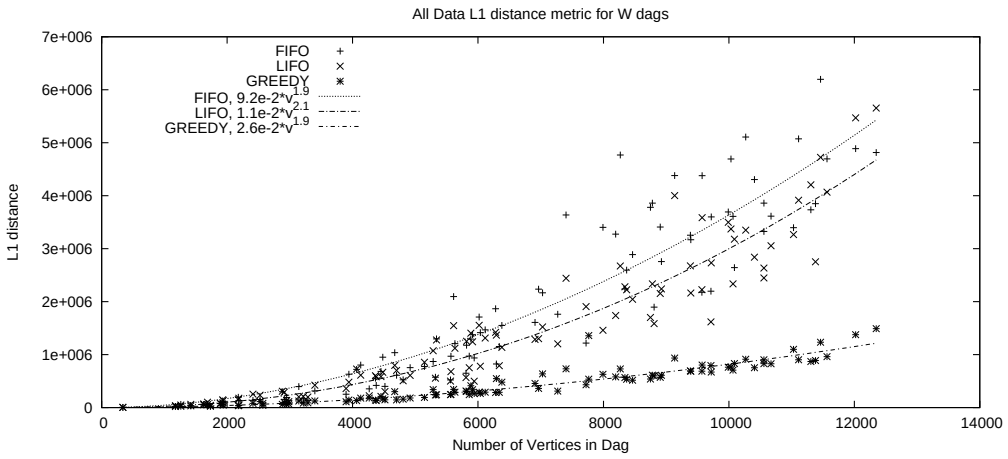

Figure 5: *Area-maximization results for random compositions of W-dags.*

5.1.2 Schedules for randomly constructed dags

As we began experimenting with randomly constructed dags, we encountered noise in the observed data that made it hard to discern trends concerning the values of $\Delta(\text{ICO}, \Sigma)$:

- for fixed Σ but different size dags,
- for different Σ's with the same dag,
- across families of kindred dags (constructed, say, from the same repertoire of CBBBs).

Fig. 5 exemplifies the problem of trying to fit a perspicuous curve to observed data. In order to filter the observed data in a way that would expose meaningful trends, we decided to eliminate all data for each heuristic Σ, except for the lower-envelope *convex hull* [6] of the values of $\Delta(\text{ICO}, \Sigma)$ across the range of generated dags. While this ploy restricted us to reporting on lower bounds on the performance disadvantage of each heuristic, it did allow us to make defensible claims about the area-maximizing advantage of the ICO scheduler, rather than to speculate on the "possible" advantage. For illustration, the graph in Fig. 6 exhibits the lower-envelope convex hull of the data from Fig. 5. We do this filtering consistently throughout this section. The W-dag-based results are supplemented in Fig. 7 with data for random compositions of, respectively, M-dags, combined W-dags, N-dags, and M-dags, and bipartite cliques. We make a few observations about our results—which should be interpreted in the light of the lower-bound nature of the data.

Comparing ICO against its competitors.

- In all the graphs presented, we observe that the functions av^b that we fit to the filtered data points all have exponents $b > 2$. This indicates that *the average per-step gain*

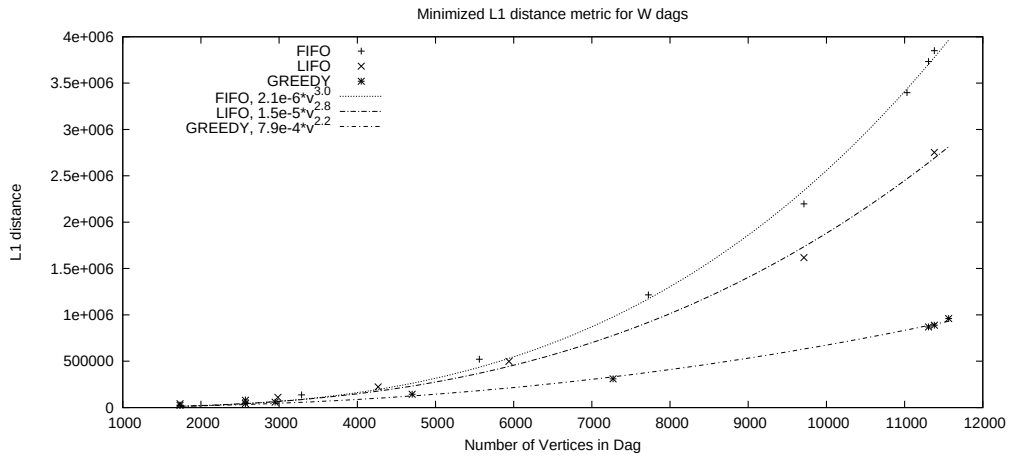

Figure 6: *Area lower bounds for random compositions of W-dags, obtained by filtering the data of Fig. 5 to show only the lower envelope.*

in ELIGIBLE *jobs from using* ICO*, rather than any of the three heuristic competitors, increases at least linearly with dag size.*

- The fact that some of the coefficients a are very small indicates that the indicated advantage is likely to be "asymptotic," taking hold only for rather large dags.

Comparisons among the competitors.

- For all generated dags, GREEDY seems to outperform both other heuristics—by a considerable margin on dags that are built using W-dags. The consistency of this observation would cause us to rank GREEDY as the best scheduler after ICO.

- FIFO *appears to be* the weakest scheduler on dags that are built using W-dags. This may be an unanticipated result of our method of ensuring IC optimal schedulability, rather than an intrinsic property of FIFO. Specifically, we always compose W-dags with smaller outdegrees "on top of" ones with larger outdegrees, which would lead FIFO to execute potentially shallow subtrees (in the expansive regions) of a dag, before executing deeper ones. Thus, one should from refrain assessing the relative strength of FIFO pending further study.

- The sparseness of data regarding random compositions of W-, N-, and M-dags weakens detailed inferences from the observed values of the parameters a and b. But, recall that these values describe lower envelopes, so all the observed data lie on or above these envelopes.

- For dags built from bipartite cliques, all three heuristic schedulers perform almost identically.

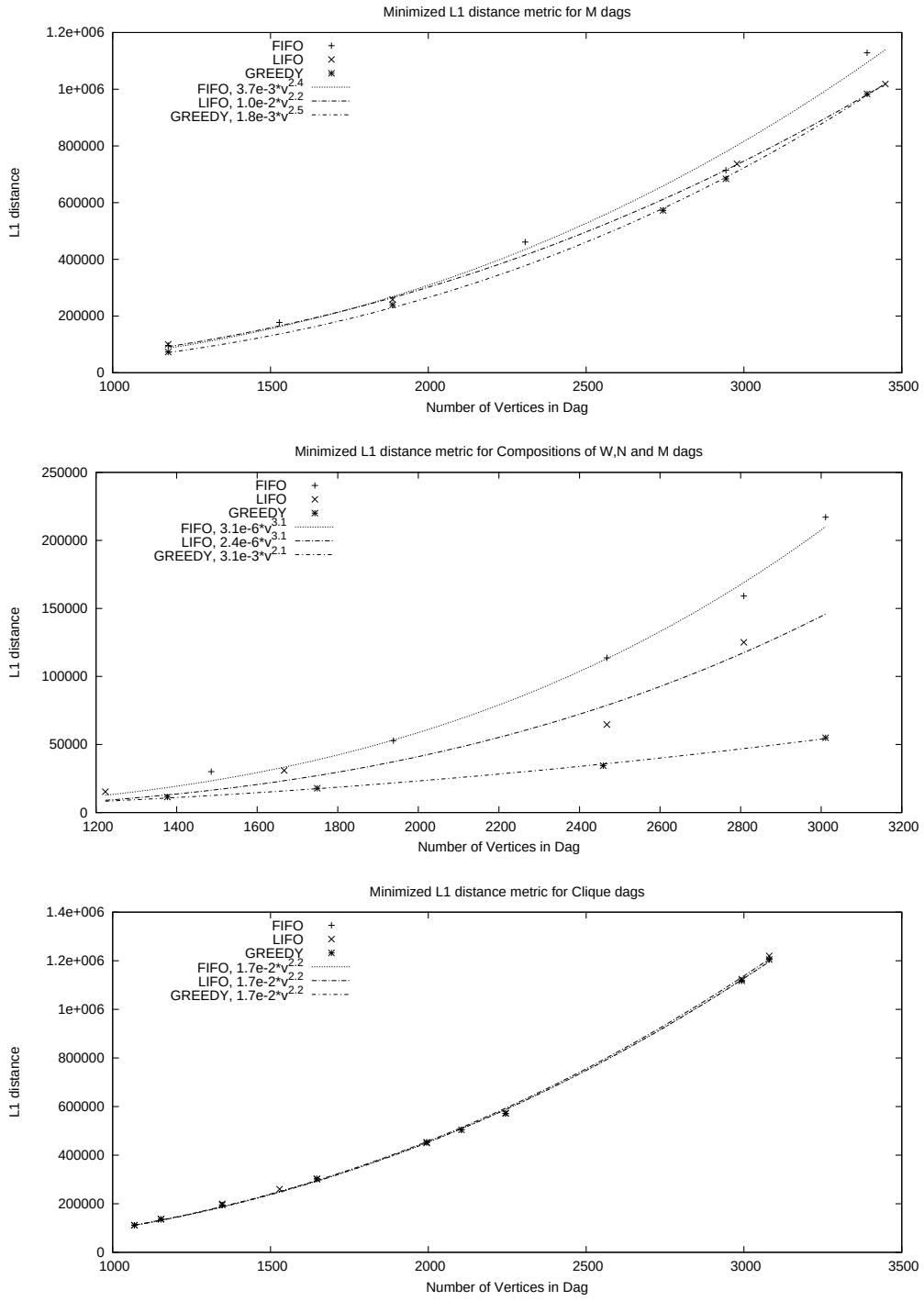

Figure 7: *Area lower bounds for random compositions of M-dags (top), combined W-, N-, and M-dags (middle), and bipartite cliques (bottom).*

5.2 Batched-Makespan Results

In all of the graphs in this section, the mean request-rate parameter ρ is plotted along the x-axis using a logarithmic scale; the phase-ratios $T(\Sigma) \div T(\text{ICO})$ appear on the y-axis. Means and 95% confidence intervals are reported for phase-ratios. Since it is difficult to display trends in these statistics over entire classes of dags, we present only two dags per class. These pairs represent trials on dags of different sizes (with the smaller on the left). We have tried to select for display dag that exhibit behavior typical of dags in their classes.

5.2.1 Schedules for familiar dags

The most notable observation from Fig. 8, wherein appear the batched-makespan results for our familiar dags, is the apparent correlation between the results obtained using the structural area-maximization metric and the behavioral batched-makespan metric. If this observation, which persists with the results concerning random dags in the next subsection, is verified by subsequent experimentation, then this could greatly simplify the scheduling problem for Internet-based computing. With our familiar dags, at least, an ordering of the four schedulers according to their relative batched-makespan performances is consistent with the analogous ordering for the area-maximization metric—at least for an important range of the request rate ρ. While we do not know how to quantify this observation, it appears to be the case qualitatively that using a scheduler that produces schedules of higher IC quality has a benign effect on the batched-makespan of the resulting schedules.

One notes in the righthand FFT graph of Fig. 8 an instance in which the phase-ratio $T(\text{GREEDY}) \div T(\text{ICO}) < 1$. This indicates that GREEDY takes fewer phases to complete than does ICO, a situation described at the end of Section 4.3.1.

5.2.2 Schedules for randomly constructed dags

Our observations concerning executing random compositions of CBBBs under the batched-makespan metric are largely qualitative. Our actual results appear in Fig. 9.

We note first that our results are quite consistent with those in [13], despite the fact that we compare ICO against three competitors, on a large range of artificially generated dags, whereas that source compares a heuristic extension of ICO against a version of FIFO, on a set of four dags that arise from actual scientific computations.

Next, we note that the overall "shapes" of the results in Figs. 8 and 9 are to be expected. No matter what class of dags is being executed, there will be only certain ranges of the client-request-rate parameter ρ for which the scheduling strategy has any impact on batched makespan. On the one hand, if requests arrive at a very slow rate, then any scheduler is likely to be able to generate enough ELIGIBLE jobs to satisfy the demand. At the other

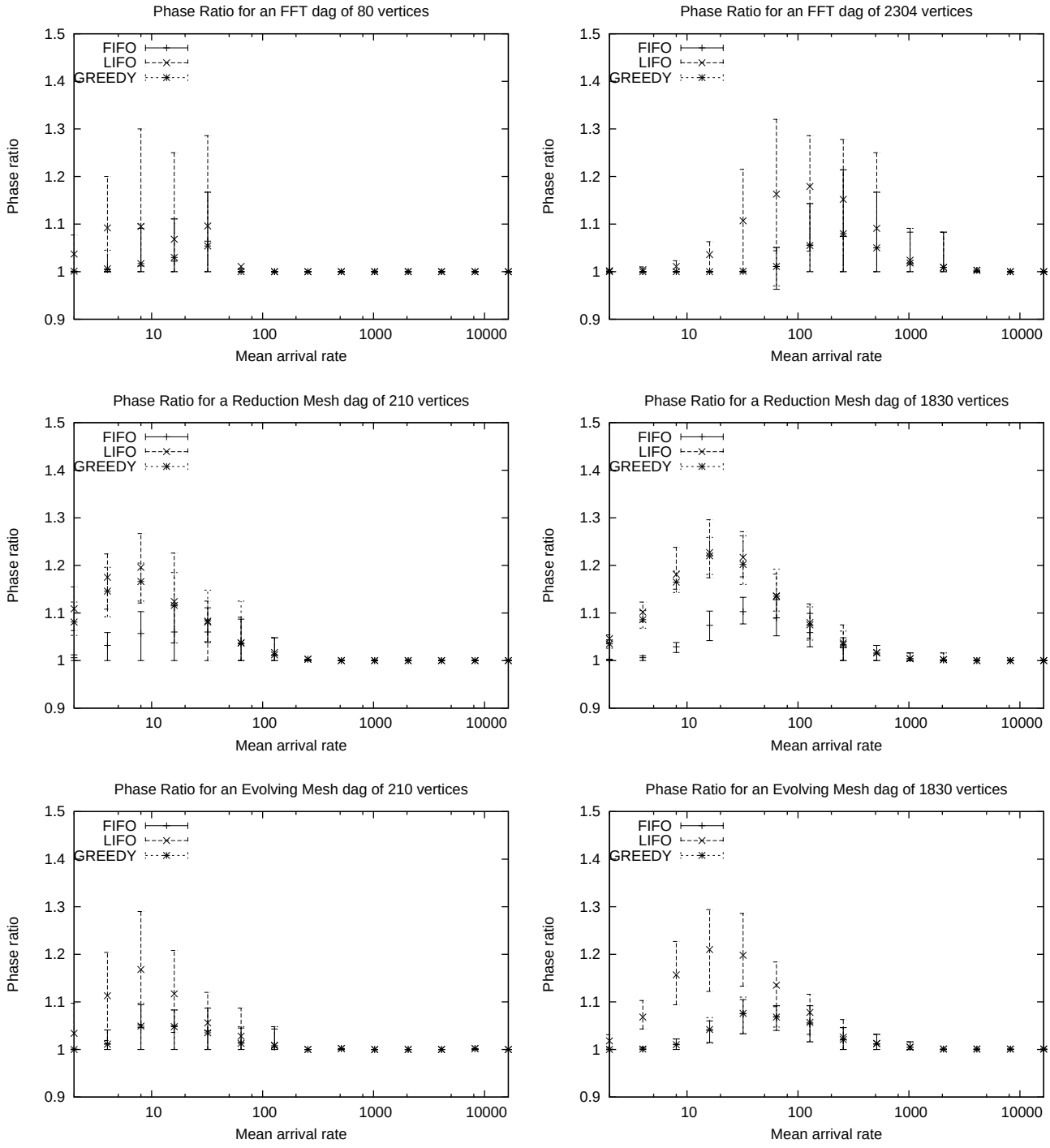

Figure 8: *Phase-ratios for two different reduction-meshes (top), evolving meshes (middle), and FFT dags (bottom).*

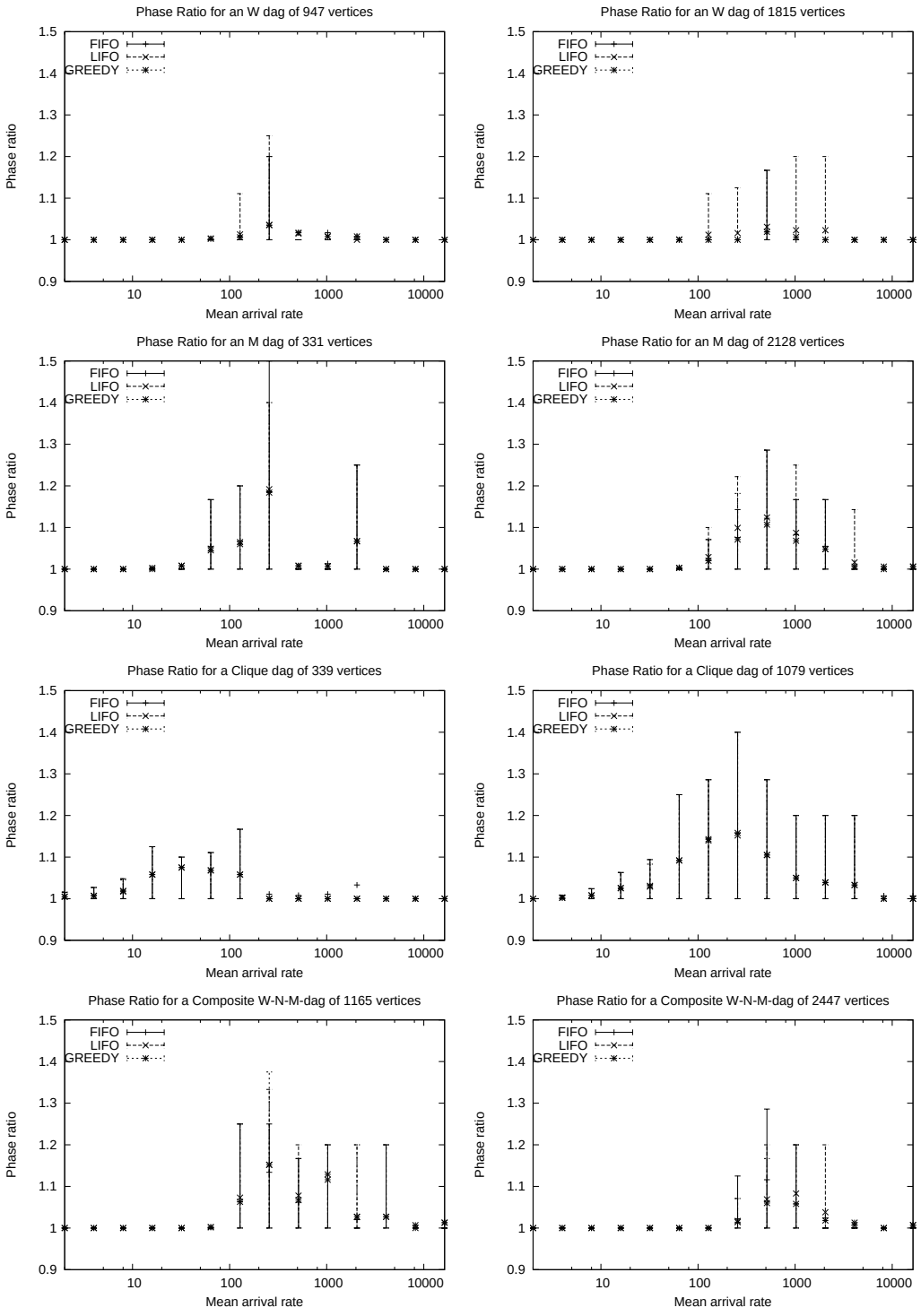

Figure 9: *From top: phase-ratios for two different compositions of: W-dags, M-dags, cliques, combined W-, N-, and M-dags.*

extreme, if there is, effectively, an unlimited supply of requests, then the batched makespan is effectively limited only by the critical path in the dag, so any approximation of a breadth-first allocation of jobs (such as FIFO and GREEDY supply) should give performance that is roughly as good as *any* scheduler can yield. It is only between these extremes that one discerns significant differences in the performances of competing schedulers.

The detailed placements and amplitudes of the "humps" depend on the structures of the dags being executed. Notably, though, in all of our experimentation, we noted only one instance of a phase-ratio dipping below 1; i.e., in essentially all experiments, ICO at least matched the batched-makespan performance of the competing heuristics. And, in many instances—see Fig. 9—ICO outperformed its competitors over a range of values of ρ by completing execution in 10–20% fewer phases than the heuristics.

6 Where We Are, and Where We're Going

Our study supplements the evidence in [13] that the nascent scheduling theory of [4, 5, 15] has significant implications for Internet-based computing. Our simulations have pitted the ICO scheduler against three natural heuristics, on hundreds of artificially generated dags, using both the area-maximization and batched-makespan quality metrics. The simulations in [13] pit an extension of ICO against a verison of FIFO, on four real scientific dags, using the batched-makespan metric. It is heartening that our results are consistent with those reported in [13].

The ultimate validation—or refutation—of the significance of the theory of IC-optimal scheduling will require actual experiments with real workloads on real computing platforms. The integration of G. Malewicz's PRIO scheduling tool into the Condor DAGMan tool [3], as described in [13], may give us this opportunity.

Acknowledgments. The research of R. Hall was performed as a "Capstone Experience" Honors thesis at UMass Amherst. The research of A. Rosenberg was supported in part by NSF Grant CCF-0342417.

References

[1] R. Buyya, D. Abramson, J. Giddy (2001): A case for economy Grid architecture for service oriented Grid computing. *10th Heterogeneous Computing Wkshp.*

[2] W. Cirne and K. Marzullo (1999): The Computational Co-Op: gathering clusters into a metacomputer. *13th Intl. Parallel Processing Symp.*, 160–166.

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

[4] G. Cordasco, G. Malewicz, A.L. Rosenberg (2006): Advances in a dag-scheduling theory for Internet-based computing. Typescript, Univ. Massachusetts. See also, On scheduling expansive and reductive dags for Internet-based computing. *26th Intl. Conf. on Distributed Computing Systems*, to appear.

[5] G. Cordasco, G. Malewicz, A.L. Rosenberg (2006): An enhanced dag-scheduling theory for Internet-based computing. Typescript, Univ. Massachusetts.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein (1999): *Introduction to Algorithms* (2nd Edition). MIT Press, Cambridge, Mass.

[7] I. Foster and C. Kesselman [eds.] (2004): *The Grid: Blueprint for a New Computing Infrastructure (2nd Edition)*. Morgan-Kaufmann, San Francisco.

[8] I. Foster, C. Kesselman, S. Tuecke (2001): The anatomy of the Grid: enabling scalable virtual organizations. *Intl. J. Supercomputer Applications*.

[9] P.R. Gill, W. Murray, M.H. Wright (1981): The Levenberg-Marquardt Method. In *Practical Optimization*. Academic Press, London, pp. 136–137.

[10] H.T. Hsu (1975): An algorithm for finding a minimal equivalent graph of a digraph. *J. ACM 22*, 11–16.

[11] D. Kondo, H. Casanova, E. Wing, F. Berman (2002): Models and scheduling mechanisms for global computing applications. *Intl. Parallel and Distr. Processing Symp.*

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

[13] G. Malewicz, I. Foster, A.L. Rosenberg, M. Wilde (2006): A tool for prioritizing DAGMan jobs and its evaluation. *15th IEEE Intl. Symp. on High-Performance Distributed Computing*, 156–167.

[14] G. Malewicz and A.L. Rosenberg (2005): On batch-scheduling dags for Internet-based computing. *Euro-Par 2005*. In *Lecture Notes in Computer Science 3648*, Springer-Verlag, Berlin, 262–271.

[15] G. Malewicz, A.L. Rosenberg, M. Yurkewych (2006): Toward a theory for scheduling dags in Internet-based computing. *IEEE Trans. Comput. 55*, 757–768.

[16] M. Mitchell (2004): Creating minimal vertex series parallel graphs from directed acyclic graphs. *Australasian Symp. on Information Visualisation.* In *Conferences in Research and Practice in Information Technology 35* (N. Churcher and C. Churcher, Eds.) ACS Press, pp. 133-139.

[17] A.L. Rosenberg (2004): On scheduling mesh-structured computations for Internet-based computing. *IEEE Trans. Comput. 53*, 1176–1186.

[18] A.L. Rosenberg and M. Yurkewych (2005): Guidelines for scheduling some common computation-dags for Internet-based computing. *IEEE Trans. Comput. 54*, 428–438.

[19] X.-H. Sun and M. Wu (2003): GHS: A performance prediction and vertex scheduling system for Grid computing. *IEEE Intl. Parallel and Distributed Processing Symp.*

[20] D. Thain, T. Tannenbaum, M. Livny (2005): Distributed computing in practice: the Condor experience. *Concurrency and Computation: Practice and Experience 17*, 323–356.