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

Robert Hall
rwhall@cs.umass.edu

Arnold L. Rosenberg
rsnbrg@cs.umass.edu

Arun Venkataramani
arun@cs.umass.edu

Univ. Massachusetts Amherst

Abstract. *A fundamental challenge in Internet computing (IC) is to efficiently schedule computations having complex interjob dependencies, given the unpredictability of remote machines, in availability and time of access. The recent IC Scheduling theory focuses on these sources of unpredictability by crafting schedules that maximize the number of executable jobs at every point in time. In this paper, we experimentally investigate the key question: does IC Scheduling yield significant positive benefits for real IC? To this end, we develop a realistic computation model to match jobs to client machines and conduct extensive simulations to compare IC-optimal schedules against popular, intuitively compelling heuristics. Our results suggest that for a large range of computation-dags, client availability patterns, and two quite different performance metrics, IC-optimal schedules significantly outperform schedules produced by popular heuristics, by as much as 10–20%.*

1 Introduction

Advances in technology have made collections of computers that communicate across the Internet a viable computational platform [5], even for solving individual computational problems [1, 2, 8]. Perhaps the major impediment to scheduling complex computations efficiently in this new environment is *temporal unpredictability*:

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

This uncertainty makes it difficult to accurately identify critical paths in complex computations, hence demands a new scheduling paradigm that acknowledges the strengths and weaknesses of the Internet as a computational medium.

Recent papers [3, 11, 12, 13] identify a new goal when scheduling computations consisting of multiple jobs with complex interdependencies for Internet-based computing (IC). These sources develop the conceptual and algorithmic foundations of *IC Scheduling* for an idealized version of IC. IC Scheduling attempts to schedule a complex computation in a manner that always maximizes the number of jobs that are eligible for allocation to remote clients, seeking to:

- utilize remote clients' computational resources well, by always having work available for allocation;
- lessen the likelihood that a computation will stall for lack of tasks that are eligible for execution.

IC Scheduling focuses on grids of committed clients (cf. the LHC Computing Grid or the UK e-Science Grid), rather than completely public ones (as in [8]). Thus, we assume that clients are trustworthy and that they may tarry but do not disappear.

IC Scheduling theory optimally schedules a large variety of common computations, such as those in Fig. 1 (whose optimal schedules are derived in [4], using algorithms from [3, 11]), as well as myriad less uniform ones. The theory seeks a regimen for scheduling complex IC computations, that has both a strong theoretical grounding and significant benefit for real computations. This paper begins to investigate the theory's benefits, via the following questions:

- What are reasonable computational models within which to evaluate the theory's performance under an *a priori* unknown sequence of available client machines?
- Does IC Scheduling theory have significant positive benefit over simpler scheduling heuristics within these models?

We address these questions by comparing schedules mandated by the theory against schedules based on popular heuristics, on randomly generated dags, using two new quality metrics. Our study shares its motivation with [9] but differs from that study in three major respects, that lead to our main contributions.

1. We test the IC-optimal scheduler, ICO, of [11] on hundreds of random dags and many client availability patterns; we generate only dags that are certain to admit IC-optimal schedules. In [9] a heuristic based on ICO is tested on four dags that arise in real computations; this heuristic produces schedules for all dags. We feel that understanding the performance of the actual ICO scheduler is a necessary test for assessing its importance for real IC.
2. We evaluate ICO against: enhanced versions of FIFO and LIFO schedulers; a greedy scheduler that employs *locally* the scheduling criteria that ICO employs *globally*; the competitor in [9] is a FIFO scheduler inspired by the one used in Condor [2].

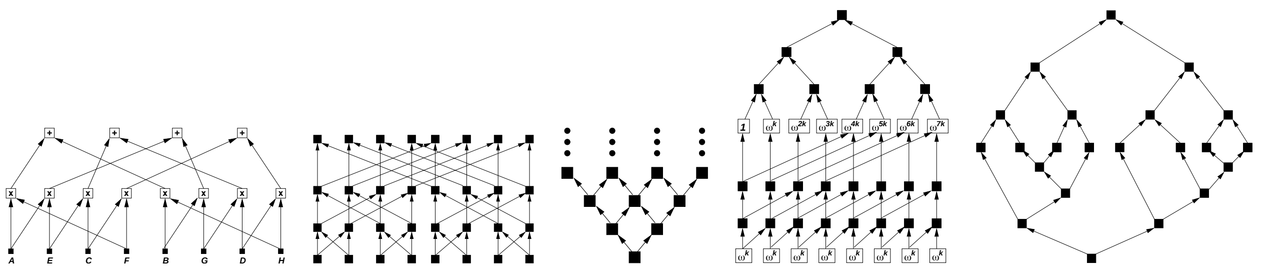

Figure 1. *Data-dependency dags for (left to right): matrix multiplication, the Fast Fourier Transform (FFT), a generic wavefront computation, the discrete Laplace transform, a generic divide-and-conquer computation.*

3. Our comparisons employ a new "area-maximizing" metric that measures the *average* rate of producing eligible jobs, in addition to the "batched makespan" metric of [10, 9].

Our results suggest: for a broad range of situations that one might expect to encounter in IC, ICO achieves significant performance improvements over its competitors for a wide range of client availability patterns.

Note 1 *We are comparing* algorithmic approaches to scheduling*, not scheduling systems that must cope with, e.g., faults, changes in the computational environment, etc.*

Related work. The most closely related study is [9], as discussed above. The closest sources involving IC Scheduling theory are: [12, 13], wherein the new scheduling paradigm is introduced and optimal schedules are computed for several uniform dags; [3, 11], wherein ICO is developed; [10], wherein our "batched makespan" quality metric is studied theoretically. Our study is motivated by the exciting systems- and/or application-oriented studies of IC in sources such as [1, 2, 5, 6, 7, 8, 14]. Traditional, critical-path-based, scheduling is not relevant to our study because temporal unpredictability renders the notion of critical path fuzzy at best.

2 Foundations of IC-Scheduling Theory

The arcs of a dag connect a *parent* node to a *child*; *sources* have no parents; *sinks* have no children.

2.1. A Quality Model for IC. When one executes a dag $\mathcal{G}$, a node $v \in N_\mathcal{G}$ is ELIGIBLE (for execution) only after all of its parents have been executed. We do not allow recomputation of nodes, so a node loses its ELIGIBLE status once it is executed. In compensation, a node v's execution may render new nodes ELIGIBLE, if v is their last parent to be executed. A *schedule* for $\mathcal{G}$ is a rule for selecting which ELIGIBLE node to execute at each step of an execution of $\mathcal{G}$. We measure the quality of an execution by the number of ELIGIBLE nodes after each node-execution—the more, the better. (Note that *we measure time in an event-driven manner*, as the number of nodes that have been executed up to that point.) Our goal is to execute $\mathcal{G}$'s nodes in an order that maximizes the production rate of ELIGIBLE nodes *at every step of the computation*. A schedule that achieves this demanding goal is IC-optimal. The significance of IC optimality stems from the following facts. (1) Schedules that produce ELIGIBLE nodes more quickly may reduce the chance of the "gridlock" that could occur when no new jobs can 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 it to satisfy more requests, thereby increasing "parallelism."

2.2. A Framework for IC-Optimal Scheduling.

One prioritizes dags as follows. 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. Let $E_{\Sigma_i}(x)$ denote the number of eligible nodes after executing x jobs using the schedule Σ_i. If the following inequalities hold:[1]

$$\begin{array}{lcl} (\forall x \in [0, s_1])\ (\forall y \in [0, s_2]) : & & \\ E_{\Sigma_1}(x) + E_{\Sigma_2}(y) & \leq & E_{\Sigma_1}(\min\{s_1, x+y\}) + \\ & & E_{\Sigma_2}(\max\{0, x+y-s_1\}) \end{array} \quad (1)$$

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

One composes dags as follows.

- Start with a set $\mathcal{B}$ of base "building block" dags (which are CBBBs[2] in [3, 11]).
- Compose dags $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{B}$—which could be the same dag with nodes 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 nodes 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 merge the nodes in S_1 and S_2 in some way. The resulting set of nodes is $\mathcal{G}$'s node-set; the induced set of arcs is $\mathcal{G}$'s arc-set.[3]

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

[2] *CBBB* is short for *C*onnected *B*ipartite *B*uilding *B*lock.

[3] An arc $(u \to v)$ is *induced* if $\{u, v\} \subseteq N_\mathcal{G}$.

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

We denote the composition operation by $\Uparrow$ and say that $\mathcal{G}$ is *composite of type* $[\mathcal{G}_1 \Uparrow \mathcal{G}_2]$.
$\mathcal{G}$ 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) $\mathcal{G}_i \triangleright \mathcal{G}_{i+1}$, for all $i \in [1, n-1]$.

Theorem 1 ([11]) *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 nodes 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 Schedulers

3.1. The IC-Optimal Scheduler ICO. One finds in [11] a suite of algorithms that determine whether or not a dag $\mathcal{G}$ can be decomposed into a set of CBBBs satisfying Theorem 1 and, if so, uses the theorem to derive an IC-optimal schedule for $\mathcal{G}$. These algorithms, collectively called scheduler ICO, process $\mathcal{G}$ via the following 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 resulting "pruned" dag $\mathcal{G}'$ shares its IC-optimal schedules with $\mathcal{G}$.
2. Decompose $\mathcal{G}'$ into CBBBs, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, such that $\mathcal{G}'$ is composite of type $\mathcal{G}_1 \Uparrow \cdots \Uparrow \mathcal{G}_n$.
   - When such a "parsing" exists, it is unique and can be found by iteratively greedily removing a maximal CBBB of $\mathcal{G}'$ all of whose sources are sources of $\mathcal{G}'$.
3. Replace $\mathcal{G}'$ by the *super-dag* $\mathcal{G}''$ whose nodes 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 sources of CBBB $\mathcal{G}_i$ with sinks of CBBB $\mathcal{G}_j$, then there is an arc in $\mathcal{G}''$ from supernode $\mathcal{G}_j$ to supernode $\mathcal{G}_i$.
4. Determine whether or not there is an $\triangleright$-linearization of $\mathcal{G}_1, \ldots, \mathcal{G}_n$ that is consistent with the topological dependencies within $\mathcal{G}''$.
   - This determines if $\triangleright$ is consistent with a topological sort of $\mathcal{G}''$.
5. If all steps have succeeded, then output the schedule for $\mathcal{G}$ mandated by Theorem 1.

3.2. The Competing Heuristic Schedulers.
A. The FIFO heuristic initially enqueues $\mathcal{G}$'s sources into a FIFO queue Q, in nonincreasing order of outdegree (maximum-outdegree nodes emerge first); nodes of equal outdegree are enqueued randomly. FIFO dequeues Q to obtain a node for a requesting client. When a node v completes execution, FIFO enqueues, in nonincreasing order of outdegree, those of v's children that are newly ELIGIBLE; nodes of equal outdegree are enqueued randomly.
B. The LIFO heuristic initially pushes $\mathcal{G}$'s sources into a (LIFO) stack S, in nondecreasing order of outdegree; nodes of equal outdegree are pushed randomly. LIFO pops S to obtain a node for a requesting client. When a node v completes execution, LIFO pushes, in nondecreasing order of outdegree, those of v's children that are newly ELIGIBLE execution; nodes of equal outdegree are pushed randomly.
C. The GREEDY heuristic initially inserts $\mathcal{G}$'s sources, in random order, into a MAX-Priority Queue P. (The ultimate order of nodes having distinct outdegrees is determined by P's queuing discipline.) GREEDY uses EXTRACT-MAX on P to obtain a node for a requesting client. When a node v completes execution, GREEDY inserts, randomly, those of v's children that are newly ELIGIBLE.

4 The Experimental Setup

We use the following experimental setup to compare ICO against FIFO, LIFO, and GREEDY.

1. We generate a dag $\mathcal{G}$ that is random within a class of dags that admit IC-optimal schedules; cf. Section 4.1.
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 [12, 13, 11]) to admit such schedules. Our selection process proceeds as follows.

1. We select a random target size for the dag we want to generate (from a few hundred nodes to several thousand).
2. We choose a collection of CBBBs randomly from a repertoire that is defined and analyzed in [11].
3. We compose the selected CBBBs in ways that are chosen randomly among compositions that are guaranteed (by Theorem 1) to preserve IC-optimal schedulability.

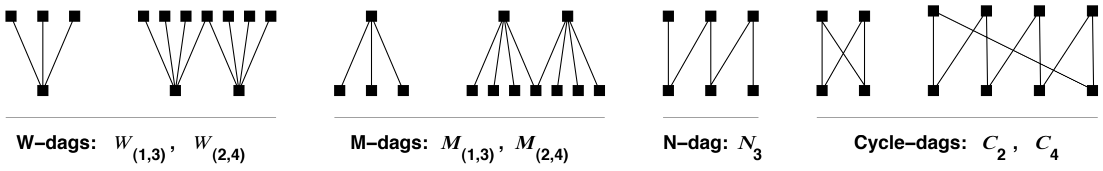
W–dags: $W_{(1,3)}$, $W_{(2,4)}$
M–dags: $M_{(1,3)}$, $M_{(2,4)}$
N–dag: N_3
Cycle–dags: C_2, C_4

Figure 2. *The CBBBs for our semi-uniform dags. Edges represent upward arcs.*

A. Selecting CBBBs. Although we select CBBBs from [11], our methodology applies easily to any CBBBs that admit IC-optimal schedules. The CBBBs we use are random-size instantiations of those depicted in Fig. 2.

B. Schedulability and $\triangleright$-priorities among the CBBBs are specified in [11].

C. Executing random composite dags. We generate dags as follows.

Selecting random CBBBs. We guarantee that generated dags admit IC-optimal schedules by constructing random $\triangleright$-linearizable compositions of CBBBs. We target dags that are likely to abstract real computations by combining CBBBs as follows, inspired by the dags in Fig. 1.

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 $\mathcal{C}_2$, abstracting convolutional dags such as the FFT dag.

Randomly composing CBBBs. Having assembled a $\triangleright$-linearizable selection of CBBBs, we composed them in a manner that is consistent with Theorem 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 selected individuals.

Executing dags. ICO is deterministic, but FIFO, LIFO, and GREEDY all employ randomness. Therefore, we had heuristics execute each generated dag fifty times and used means and deviations from the results in our comparisons.

4.2. The Area-Maximization Experiment.

A. The area-maximization metric. IC optimality rewards a schedule Σ for maximizing the number of ELIGIBLE nodes *at every step* while executing a dag $\mathcal{G}$. The *area-maximization* metric rewards Σ for maximizing the *average* number of nodes of $\mathcal{G}$ that are ELIGIBLE as $\mathcal{G}$ is executed. We term this average the "area" of Σ, because of the formal analogy with Riemann sums as approximate integrals.

The *plot* of schedule Σ is the $(n+1)$-entry vector $\Pi(\Sigma) = \langle E_\Sigma(0),\ E_\Sigma(1), \ldots,\ E_\Sigma(n)\rangle$. (We retain entries $E_\Sigma(0) =$ the number of sources of $\mathcal{G}$, and $E_\Sigma(n) \equiv 0$ for completeness.) The *area* of schedule Σ is $A(\Sigma) = \sum_{i=0}^{n} E_\Sigma(i)$. Of course, the normalized area $\widehat{E}(\Sigma) \stackrel{\text{def}}{=} \frac{1}{n}A(\Sigma)$ is the average number of nodes of $\mathcal{G}$ that are ELIGIBLE under schedule Σ.

Notes. **(a)** *Some dags do not admit any IC-optimal schedule [11], but every dag admits an area-maximizing schedule.* **(b)** *If a dag $\mathcal{G}$ admits an IC-optimal schedule, then every area-maximizing schedule for $\mathcal{G}$ is IC optimal, and every IC-optimal schedule for $\mathcal{G}$ is area-maximizing.*

B. The area-maximization experiment. This experiment generates random dags in the manner described earlier. We study each generated dag $\mathcal{G}$ as follows.

1. We compute $\widehat{E}(\text{ICO})$ directly, as $\mathcal{G}$ is generated.
2. For each heuristic Σ, we execute $\mathcal{G}$ fifty times and compute the mean of $\widehat{E}(\Sigma)$, denoted $\widetilde{E}(\Sigma)$, and the standard deviation.

We compare schedulers Σ and Σ' under this metric via the quantity $\Delta(\Sigma, \Sigma') \stackrel{\text{def}}{=} \widetilde{E}(\Sigma) - \widetilde{E}(\Sigma')$. [$\widetilde{E}(\text{ICO}) \equiv \widehat{E}(\text{ICO})$ by convention.] (Note: $n \cdot \Delta(\Sigma, \Sigma')$ is the $L1$ distance between $\Pi(\Sigma)$ and $\Pi(\Sigma')$.) As just observed, $\Delta(\text{ICO}, \Sigma') \geq 0$.

4.3. The Batched-Makespan Experiment.

A. The batched-makespan metric. This quality metric compares schedulers using a "server-centric," rather than "client-centric," model of IC. The *"client-centric" model*—which is the one studied in [3, 11, 12, 13]—views the Server as being interrupted by the arrival of an available remote client. In response, the Server allocates an ELIGIBLE job to the client, if one exists; otherwise, the client "disappears" (say, looking elsewhere for work). The *"server-centric" model*—a variant of the model in [10]—has remote clients arrive in groups at preassigned times—perhaps, but not necessarily, periodically. At these times, the Server polls for the presence of both clients and ELIGIBLE jobs. When a poll finds, say, $r \geq 1$ remote clients and $e \geq 1$ ELIGIBLE jobs, the Server chooses $\min(r, e)$ ELIGIBLE jobs and allocates them, one per client, until either clients or jobs run out. At this point, unserved clients "disappear" and unallocated jobs are returned to the ELIGIBLE pool. The "server-centric" model suggests the *batched-makespan metric* for a scheduler Σ when executing an n-node dag $\mathcal{G}$. Given a pattern of arrivals of batched requests, $r_0, r_1, \ldots$, meaning that, for each i, r_i clients arrive requesting jobs at the Server's ith poll, how many pollings does it take to execute $\mathcal{G}$? Letting $E'_\Sigma(i)$ denote the number of ELIGIBLE jobs at the ith polling, *we seek the smallest integer m such that $a_0 + a_1 + \cdots + a_m \geq n$, where $a_0 = \min(r_0, E'_\Sigma(1))$, $a_1 = \min(r_1, E'_\Sigma(2)), \ldots,\ a_m = \min(r_m, E'_\Sigma(m))$.* Under this model, the Server may have to allocate $r > 1$ ELIGIBLE jobs at once at some 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 apparent anomaly: *Under the batched-makespan metric, some schedulers can conceivably outperform scheduler* ICO *on some dags*. Thus, under this metric, ICO is actually a heuristic. We justify using a heuristic by noting that optimizing even a single step under this metric is NP-Complete [10].

B. The batched-makespan experiment. This experiment generates random dags as described earlier and studies the execution of each dag as follows. We have the Server poll for clients requesting work according to an externally specified schedule. At each poll, the number ρ of requests for work is a random variable with values distributed exponentially in the set $[2, 2^{14}]$. (Thus, each polling is independent of all others.) In common with [9], we assume that job execution times are distributed normally, with mean 1 and standard deviation 0.1. The variability in the sizes of generated dags, our range of values for ρ, and the assumed variability in job execution times combine to give us a picture of how our four schedulers behave under a rather broad range of situations.

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*, with $T(\Sigma)$ ($\Sigma \in \{\text{ICO}, \text{FIFO}, \text{LIFO}, \text{GREEDY}\}$ denoting the mean observed number of pollings required by scheduler Σ. For each $\mathcal{G}$, we compare ICO against its three competitors via the *phase-ratio* $T(\Sigma) \div T(\text{ICO})$ (so larger ratios favor ICO).

5 Experimental Results and Interpretation

5.1 Area-Maximization Results. We present both means and 95% confidence intervals for $\Delta(\text{ICO}, \Sigma)$, for $\Sigma \in \{\text{FIFO}, \text{LIFO}, \text{GREEDY}\}$. (The intervals are often so tight that they are indistinguishable from the means.) To be conservative and perspicuous, we fitted curves of the form $a \cdot v^b$ (v is the size of the generated dag) to the convex hull of the lower envelope of the observed data. Thus fit, $\Delta(\text{ICO}, \Sigma)$ always grew superlinearly and often few super-quadratically with dag size!

A. Familiar dags. We instantiated the dags of Fig. 1 in several different sizes: 3-to-10-level FFT dags and 10-to-100-level mesh-like dags (to equalize the sizes of the largest dags tested). The plots in Fig. 3 expose a number of meaningful patterns. Most importantly, $\Delta(\text{ICO}, \Sigma)$ grows nontrivially with the size of the dag being scheduled. Specifically, *excepting evolving meshes (and evolving trees), which (almost) any strategy schedules well, we found that:*

> $\Delta(\text{ICO}, \Sigma)$ *grows* $\underline{\textit{superlinearly}}$ *with dag size.*
> so that:
> *The average per-step gain in* ELIGIBLE *jobs from using* ICO *rather than a heuristic grows with dag size.*

B. Random composite dags. When perusing our plots in Fig. 4, and the following observations, keep in mind that we are discussing *lower bounds* on $\Delta(\text{ICO}, \Sigma)$.

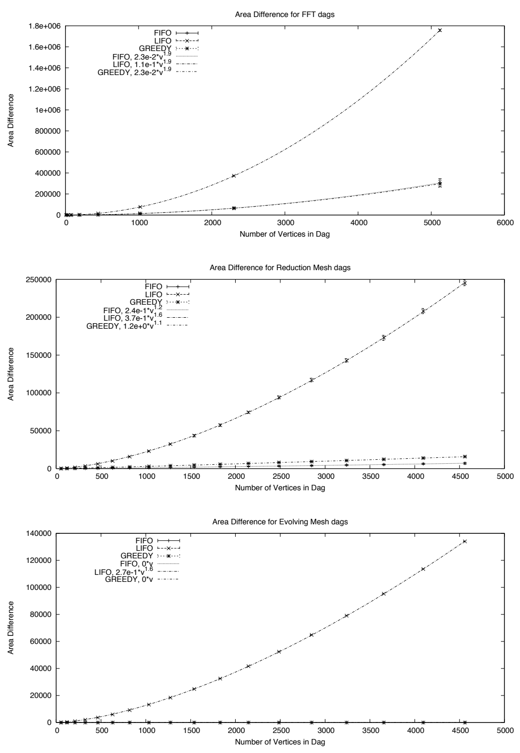

Figure 3. *Area-maximization results for FFT dags (top), reduction-meshes (middle), and evolving meshes (bottom).*

Comparing ICO *against its competitors.*

> $\Delta(\text{ICO}, \Sigma)$ *grows* $\underline{\textit{superquadratically}}$ *with dag size.*
> so that:
> *The average per-step gain in* ELIGIBLE *jobs from using* ICO *rather than a heuristic grows* $\underline{\textit{superlinearly}}$ *with dag size.*

The fact that some coefficients a are very small suggests that the indicated advantage may be discernible only for rather large dags.

Comparisons among the competitors.

- GREEDY consistently outperforms both other heuristics—by a considerable margin on compositions of W-dags. This suggests that GREEDY is the best heuristic scheduler.
- FIFO *appears to be* the weakest scheduler on compositions of W-dags. This *may* result from our composition regimen, which always places W-dags with smaller outdegrees "on top of" ones with larger outdegrees, which leads FIFO to execute potentially shallow subtrees in the expansive regions of a dag, before deeper ones. Thus, we cannot yet reliably assess the relative quality of FIFO's schedules.
- The sparseness of data regarding random compositions

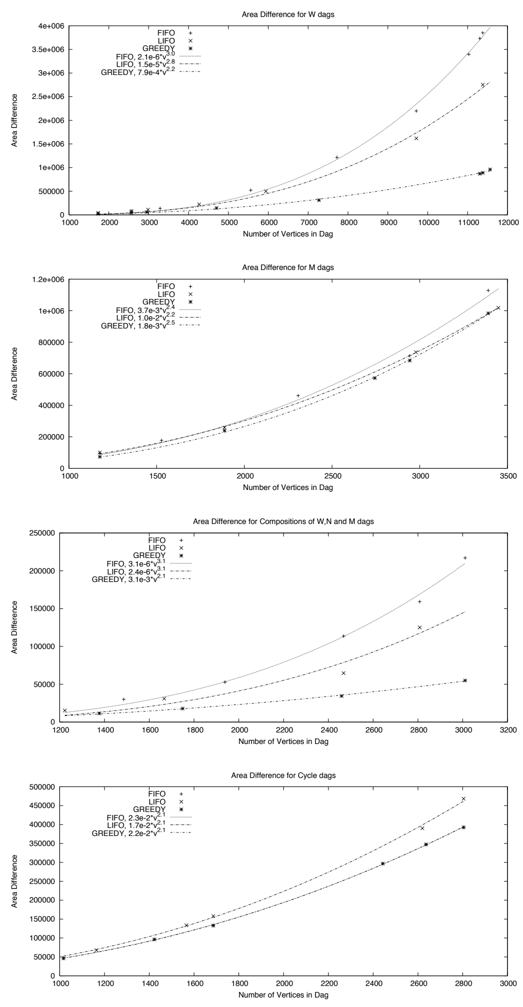

Figure 4. *Area-maximization results for random compositions of W-dags (top), M-dags (second), combined W-, N-, and M-dags (third), and bipartite cycles (bottom).*

of W-, N-, and M-dags weakens detailed inferences from the observed values of a and b. But, recall that these values describe only lower envelopes.

- The heuristic schedulers perform almost identically on dags built from cycles.

5.2. Batched-Makespan Results. The difficulty of crafting perspicious 2-dimensional illustrations of our batched-makespan results, led us to present data here for only two dags from each class that we studied, selecting dags whose results are typical of those observed throughout the class. The selected results compare the batched-makespan performance of each heuristic Σ, as compared with that of ICO, by plotting the phase-ratio $T(\Sigma) \div T(\text{ICO})$, represented via means and 95% confidence intervals, as a function of the *client arrival rate* ρ (plotted in a logarithmic scale along the x-axes of the plots). Our results suggest an unexpected consistency between the *structural* area-maximization metric and the *behavioral* batched-makespan metric—at least for an important range of values of ρ. Specifically, it appears that using schedules of higher IC quality has a benign effect on batched-makespan. If this observation is verified by subsequent (planned) study, then this could greatly simplify the scheduling problem for IC.

A. Familiar dags. The FFT-dag plots of Fig. 5 contain instances in which $T(\text{GREEDY}) \div T(\text{ICO}) < 1$, indicating that GREEDY sometimes takes fewer phases to complete than does ICO, a situation described at the end of Section 4.3.1.

B. Random composite dags. The plots in Fig. 6 provide insight into schedulers' "random performance" under the batched-makespan metric. Notably, these results are quite consistent with those in [9], despite the differences in the two studies, as described earlier. The overall "shapes" of the plots in Figs. 5 and 6 are expected. For extreme values of ρ, scheduling strategy has no impact on batched makespan. If requests are very sparse, then any scheduler will generate enough ELIGIBLE jobs. If there is, effectively, an unlimited supply of requests, then the batched makespan is really limited only by the sequential depth of the dag, so any approximately breadth-first allocation of jobs should be roughly as good as any other. It is only between these extremes that one discerns significant differences among schedulers. The detailed placement and amplitude of the "humps" in the plots depend on the structure of the dag being executed. Notably, though, in no experiment did we note a *mean* phase-ratio below 1; i.e.: **in all experiments, ICO at least matched the batched-makespan performance of the competing heuristics.** And, in many instances—cf. Fig. 6—ICO completed execution in 10-20% fewer phases than its competitors, over a range of values of ρ.

6 Where We Are, and Where We're Going

Our study supplements the evidence in [9] that IC-Scheduling theory has significant postive implications for Internet computing. Our simulations have pitted the ICO scheduler against three natural heuristics, on hundreds of randomly generated dags, using both the area-maximization and batched-makespan quality metrics. The simulations in [9] pit an extension of ICO against a verison of FIFO on four real dags, using the batched-makespan metric. The consistency between our results and those in [9] strengthens our confidence in the new theory. Of course, the ultimate validation—or refutation—of the significance of IC Scheduling theory will require experiments with real workloads on real computing platforms. The integration of G. Malewicz's PRIO scheduling tool into the Condor DAGMan tool [2] (cf. [9]) may give us this opportunity. This paper contributes one more step toward confirming the value of such an endeavor.

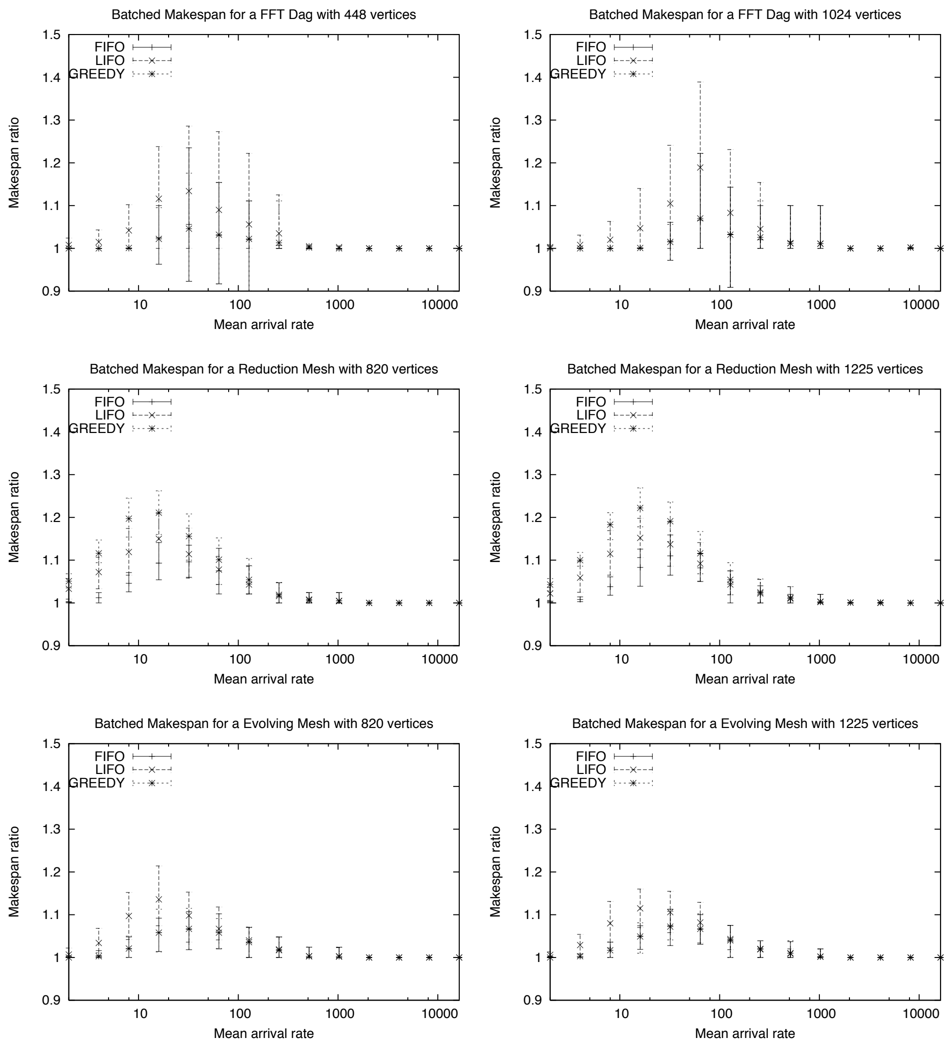

Figure 5. *Phase-ratios for two different FFT dags (top), reduction meshes (middle), and evolving meshes (bottom).*

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

[3] G. Cordasco, G. Malewicz, A.L. Rosenberg (2007): Advances in IC-scheduling theory: scheduling expansive and reductive dags and scheduling dags via duality. *IEEE Trans. Parallel and Distributed Systems*, to appear.

[4] G. Cordasco, G. Malewicz, A.L. Rosenberg (2007): Applying IC-scheduling theory to familiar classes of computations. *Wkshp. on Large-Scale, Volatile Desktop Grids (PCGrid'07)*, to appear.

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

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

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

[8] 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.

[9] 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 Distr. Computing*, 156–167.

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

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

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

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

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

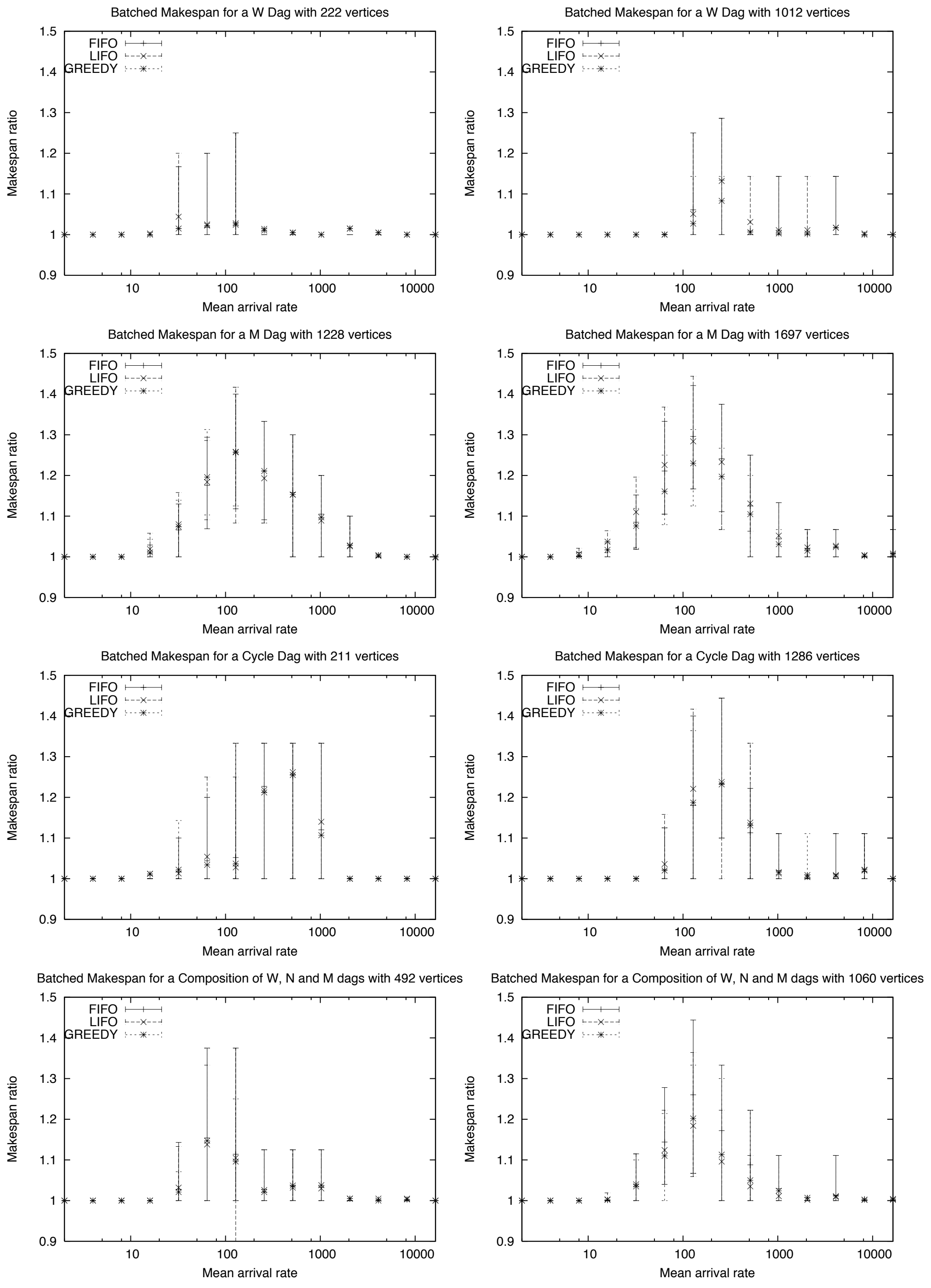

Figure 6. *Phase-ratios for two compositions of (from top): W-dags, M-dags, cycles, combined W-, N-, and M-dags.*