Scheduling DAGs Opportunistically: The Dream and the Reality Circa 2016

Arnold L. Rosenberg(✉)

Computer Science, Northeastern University, Boston, MA, USA
rsnbrg@ccs.neu.edu

Abstract. A broad-brush tour of a platform-oblivious approach to scheduling DAG-structured computations on platforms whose resources can change dynamically, both in availability and efficiency. The main focus is on the IC-scheduling and Area-oriented scheduling paradigms—the motivation, the dream, the implementation, and initial work on evaluation.

Keywords: Area-oriented DAG-scheduling · Dynamically changing platforms · IC-DAG-scheduling · Opportunistic DAG-scheduling

1 Prehistory

Early this century, Fran Berman, then-director of the San Diego Supercomputing Center (SDSC), gave a distinguished lecture at my then-home institution, UMass-Amherst. During a subsequent one-on-one, Fran educated me about a Grid-consortium that SDSC participated in, jointly with several kindred centers. The consortium "contract" allowed any member institution to submit computing jobs to any other. There was a guarantee that submitted jobs would be completed—but not when. When I asked what kind of computations SDSC performed using this paradigm, I was shocked to learn that the computations had dependencies among subcomputations that constrained the order in which work could be done. (As I recall, these were wavefont-structured dependencies.) I asked Fran how her team coped with the possibility that work could grind to a halt pending the completion of jobs that had been deployed within the consortium but not yet completed. Fran responded that they used heuristics that seemed to work well—but that she did not know of any mathematical setting that would allow one to think about this situation rigorously. The challenge was irresistible!

2 The Dream of Opportunistic Scheduling

2.1 An Informal Overview

Many modern computing platforms—notably including clouds [26,27], desktop grids [2], and volunteer-computing projects [11,15]—exhibit extreme levels of

© Springer International Publishing Switzerland 2016
P.-F. Dutot and D. Trystram (Eds.): Euro-Par 2016, LNCS 9833, pp. 22–33, 2016.
DOI: 10.1007/978-3-319-43659-3_2

dynamic heterogeneity. The availability and relative efficiencies of such platforms' computing resources can change at unexpected times and in unexpected ways. Scheduling a computation for efficient execution on such a platform can be quite challenging, particularly when there are dependencies among the computation's constituent *chores*[1] (jobs, tasks, etc.). We wanted to take up this challenge for the traditional scheduling setting of computations whose dependencies had the structure of DAGs (*directed acyclic graphs*).

The nodes of a computation-DAG $\mathcal{G}$ represent chores to be executed; $\mathcal{G}$'s arcs (directed edges) represent inter-chore dependencies that constrain the order in which chores can be executed. Specifically, a node v cannot be executed until all of its *parents* have been: these are the nodes that have arcs *into* v. Once all of v's parents have been executed, v becomes *eligible* (for execution) and remains so until it is executed. $\mathcal{G}$ has one or more *sources*—nodes that have no parents, hence are immediately eligible—and one or more *sinks*—nodes that have no "children." Clearly, executing a non-sink renders new nodes eligible. The execution of $\mathcal{G}$ terminates once all nodes have been executed.

2.2 Opportunistic DAG-Execution via Platform-Oblivious Scheduling

Recent studies have proposed seeking high performance and low cost within platforms that are dynamically heterogeneous and/or elastic by scheduling computations in a *platform-oblivious* manner. One compensates for ignoring platform details by carefully exploiting the detailed characteristics of one's computation. The central thesis motivating this approach is that, particularly with the targeted platforms, one always benefits computationally with DAG-structured workflows *by enhancing the likelihood of having as many eligible chores as possible*. Such scheduling enhances the likelihood of having work available as (advantageous) resources become available, hence being able to exploit resources *opportunistically*. Platform-oblivious scheduling can be advantageous for the targeted platforms because it exploits unchanging, perfectly-known characteristics of one's computation rather than attempting to adapt to characteristics of the platform, which are at best imperfectly known and, indeed, may change dynamically.

As we have pursued the dream of high-performing platform-oblivious schedules, we have found it technically advantageous to follow the lead of work-centric systems such as CHARM++ [14], by refining input DAGs before scheduling. We thereby can focus on scheduling *fine-grained* DAGs whose chores are all of (roughly) equal complexity. This focus extrapolates easily to DAGs that represent heterogeneous workloads: one simply models large chores as chains of "unit-size" ones with sequential dependencies, in the manner discussed in [9].

3 The Reality

3.1 Formalizing the Dream

A schedule Σ for a DAG $\mathcal{G}$ is a *topological sort* [10] of $\mathcal{G}$, i.e., a linear ordering of $\mathcal{G}$'s nodes in which all parents of each node v lie to the left of v. The schedule prescribes

[1] We use the granularity-neutral "chore" for the units that form the computation.

the order in which $\mathcal{G}$'s nodes are selected for execution. For any schedule Σ for $\mathcal{G}$ and any integer $T \in [0..N_{\mathcal{G}}]$,[2] $E_{\Sigma}(T)$ denotes the number of nodes of $\mathcal{G}$ that are eligible for execution at step T when Σ executes $\mathcal{G}$.

A. ICO Quality and Optimality [21]. Our first quality measure for DAG-schedules embodies the strictest possible interpretation of "eligible-node enhancement." We measure the *IC quality* of an execution of $\mathcal{G}$ by the number of nodes that are eligible 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 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 execution*, i.e., to craft a schedule $\Sigma^{\star}$ such that

$$(\forall t) \quad E_{\Sigma^{\star}}(t) \;=\; \max_{\Sigma \text{ a schedule for } \mathcal{G}} \{E_{\Sigma}(t)\}. \tag{1}$$

A schedule for $\mathcal{G}$ that achieves this demanding goal is *IC optimal* (*ICO*, for short).

In Sect. 3.2.A, we discuss ICO schedules for many classes of significant "real" computations—surprisingly many, given the strictness of the condition in Eq. 1.

B. AREA Quality and Optimality [3]. As we detail in Sect. 3.2.A, the demands of Eq. 1 are so stringent that many DAGs do not admit ICO schedules. This led us to weaken the IC-scheduling paradigm in [3], by introducing the *Area-oriented* DAG-scheduling paradigm.

Let Σ be a schedule for DAG $\mathcal{G}$. The *Area*, $Area(\Sigma)$, of Σ, is the sum

$$Area(\Sigma) \;=\; E_{\Sigma}(0) + E_{\Sigma}(1) + \cdots + E_{\Sigma}(N_{\mathcal{G}}).$$

Note that schedule Σ's normalized Area—obtained by dividing $AREA(\Sigma)$ by the number of nodes in $\mathcal{G}$—is the average number of nodes that are eligible as Σ executes $\mathcal{G}$. (The term *Area* is by analogy with Riemann sums approximating integrals.) Our goal is to find, for each DAG $\mathcal{G}$, an Area-maximal schedule, i.e., a schedule $\Sigma^{\star}$ for $\mathcal{G}$ such that

$$Area(\Sigma^{\star}) \;=\; \max_{\Sigma \text{ a schedule for } \mathcal{G}} Area(\Sigma). \tag{2}$$

A schedule for $\mathcal{G}$ that achieves this goal is *Area-optimal* (*A-O*, for short).

Easily, every DAG admits an A-O schedule. Importantly for our dream, the A-O scheduling paradigm is a strict extension of the ICO paradigm, in the following sense.

Theorem 1 ([3]). *If* DAG $\mathcal{G}$ *admits an ICO schedule* Σ*, then every ICO schedule for* $\mathcal{G}$ *is A-O, and vice versa.*

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

C. Optimal Schedules via DAG-duality. An important "meta-scheduling" contribution appears in [6] for ICO scheduling and in [3] for A-O scheduling. In both cases, one finds an algorithm that converts an optimal ICO (resp., A-O) schedule for a DAG $\mathcal{G}$ to an optimal ICO (resp., A-O) schedule for $\mathcal{G}$'s *dual* DAG $\widehat{\mathcal{G}}$. $\widehat{\mathcal{G}}$ is obtained from $\mathcal{G}$ by reversing all of $\mathcal{G}$'s arcs (e.g., the evolving mesh and reduction-mesh in Fig. 1(a) are dual to each other, as are the expansion-tree and reduction-tree in Fig. 1(b)).

3.2 Finding High-Quality Schedules

A. Schedules with High ICO Quality. The stringent demands of IC-*optimality*—the *maximum* number of eligible nodes at *every* step of a DAG-execution; cf. Eq. 1—raises the specter that ICO schedules exist only for a very constrained class of DAGs. Our first goal was to refute this possibility. We derived the following results.

(1) *ICO schedules for specific families of* DAG*s and computations.* In [6,21,22], we developed ICO scheduling strategies for many familiar classes of DAGs, including

- *evolving meshes* and *reduction-meshes*; see Fig. 1(a)
- *expansion-trees* and *reduction-trees*; see Fig. 1(b)
- *butterfly-structured, convolutional* DAG*s*; see Fig. 1(c, right).

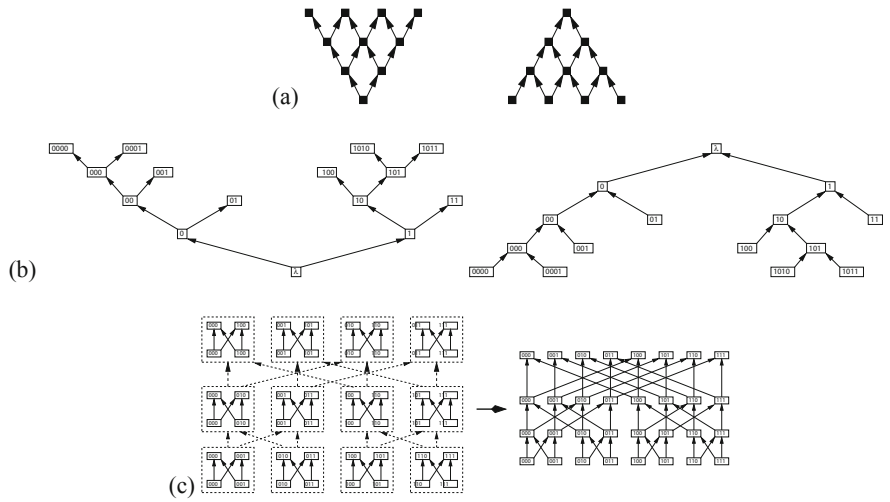


Fig. 1. Familiar DAGs that admit ICO schedules

In [5], we expanded the abstract, DAG-oriented, perspective of the preceding sources, to develop ICO scheduling strategies for many familiar classes of *computations*, including

– *convolutions*—e.g., the Fast Fourier Transform, polynomial multiplication
– *expansion-reductions*—e.g., numerical integration, comparator-based sorting
– many "named" computations—e.g., *Discrete Laplace Transform*, *matrix multiply*.

(2) *ICO schedules via* DAG *decomposition.* Careful analysis of our *ad hoc* schedules enabled us, in [19], to develop efficient—i.e., quadratic-time—algorithms that produce ICO schedules for a broad range of DAGs, based on structural decomposition. When the strategy succeeds in decomposing a DAG $\mathcal{G}$ in the prescribed manner, one can "read off" an ICO schedule for $\mathcal{G}$ from the decomposition. The strategy has two major steps.

Step 1. Select a set of bipartite[3] "building-block" DAGs that admit ICO schedules.

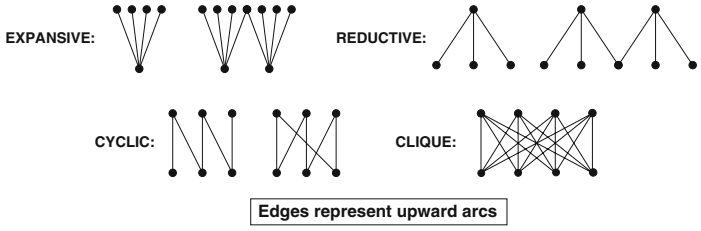


Fig. 2. A sampler of small instances of useful bipartite "building-block" DAGs.

The chosen "building blocks" will be the atomic computations in the schedule. The sample repertoire in Fig. 2 fits both needs that are salient for our strategy. (*a*) The illustrated DAGs are reminiscent of pieces of the interchore dependency-DAGs for a broad range of significant computations. (*b*) These DAGs admit ICO schedules. Indeed, *any schedule for these* DAG*s that executes all sources sequentially is an ICO schedule.*

Step 2. Establish $\triangleright$-priorities among the building-block DAGs.
For $i = 1, 2$, let DAG $\mathcal{G}_i$ admit an IC-optimal schedule Σ_i. We say that $\mathcal{G}_1$ has $\triangleright$-*priority* over $\mathcal{G}_2$—denoted $\underline{\mathcal{G}_1 \triangleright \mathcal{G}_2}$—precisely when the following recipe produces an ICO schedule for executing both $\mathcal{G}_1$ and $\mathcal{G}_2$ (i.e., for executing the *sum* of $\mathcal{G}_1$ and $\mathcal{G}_2$):

First: Execute $\mathcal{G}_1$ by following schedule Σ_1.

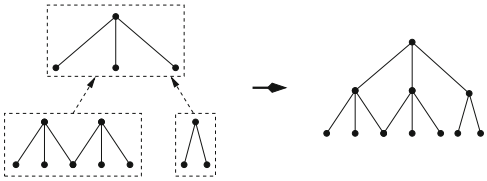
Then: Execute $\mathcal{G}_2$ by following schedule Σ_2.

One verifies that *relation* $\triangleright$ *is transitive and efficiently tested* [7].

The next ingredient in our strategy focuses on creating complex computation-DAGs by *composing* simple computation-DAGs. One *composes* DAG $\mathcal{G}_1$ with DAG $\mathcal{G}_2$ by *merging/identifying* some k sources of $\mathcal{G}_2$ with some k sinks of $\mathcal{G}_1$: the resulting DAG is *composite of type* $\mathcal{G}_1 \Uparrow \mathcal{G}_2$. (Easily, DAG-composition composes

[3] A bipartite DAG's nodes are partitioned into sets X and Y, with every arc going from X to Y.

the function specified by $\mathcal{G}_1$ with the one specified by $\mathcal{G}_2$.) The following sample composition illustrates DAG-composition and its associativity.



We can now announce the major contribution of our decomposition-based strategy.

Theorem 2 ([19]). *Focus on a* DAG $\mathcal{G}$ *that is composite of type* $\mathcal{G}_1 \Uparrow \mathcal{G}_2 \Uparrow \cdots \Uparrow \mathcal{G}_n$. *Say that*

— *each* DAG $\mathcal{G}_i$ *admits the IC-optimal schedule* Σ_i*;*

— $\mathcal{G}_1 \rhd \mathcal{G}_2 \rhd \cdots \rhd \mathcal{G}_n$.

Then, the following schedule for $\mathcal{G}$ *is IC optimal:*

Use the schedules $\{\Sigma_i\}$ *to execute the* DAGs $\{\mathcal{G}_i\}$ *seriatim, in order of* $\rhd$*-priority.*

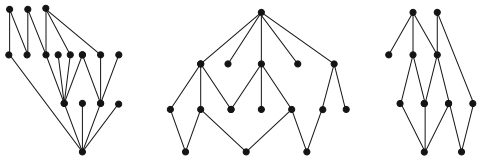
Efficient algorithms implement Theorem 2 on a large variety of "well-structured" DAGs. In particular, the two core processes in the theorem are computationally efficient:

— "parsing" DAG $\mathcal{G}$ into $\mathcal{G}_1, \ldots, \mathcal{G}_n$ (when such a parsing exists)

— testing $\rhd$-priorities among the $\mathcal{G}_i$.

Two clarifications will help illuminate Theorem 2.

1. *A* DAG *can have very nonlinear structure, even though it is composed from small* DAG*s that obey a linear chain of* $\rhd$*-priorities.*
   Butterfly DAGs provide an example. Every butterfly DAG $\mathcal{B}$ is composed from many copies of the bipartite butterfly DAG $\mathcal{B}_2$: symbolically, $\mathcal{B}$ is composite of type $\mathcal{B}_2 \Uparrow \mathcal{B}_2 \Uparrow \cdots \Uparrow \mathcal{B}_2$ (see Fig. 1(c)). One verifies easily that $\mathcal{B}_2$ has "self $\rhd$-priority"—i.e., $\mathcal{B}_2 \rhd \mathcal{B}_2$—so that $\mathcal{B}$ admits a linear chain of $\rhd$-priorities: $\mathcal{B}_2 \rhd \cdots \rhd \mathcal{B}_2$.
2. *Many* DAG*s that admit ICO schedules are quite nonuniform in a graph-structure sense:*



   The "well-structuredness" exploited in Theorem 2 is algebraic in nature, in terms of composition and $\rhd$-priority.

(3) *A weakness in the IC-scheduling paradigm.* Using Theorem 2 and *ad hoc* techniques, we developed ICO—i.e., *optimal* eligible-node-enhancing—schedules for many popular families of DAGs, including "butterflies," "meshes," "trees."

But, with little difficulty, we also discovered "cousins" of these "well-structured" DAGs that *do not admit any ICO schedule* [19]. This deficiency in the IC-scheduling paradigm—the existence of unoptimizable schedules—led us to seek "weakened" versions of the paradigm that would algorithmically produce schedules for *every* input DAG, that were optimizable according to a quality metric that correlated with computational performance. We discovered two such paradigms.

1. A *batched* notion of ICO quality is introduced in [17]. The underlying idea is to execute a DAG by choosing successive *subsets* of the then-eligible nodes.
2. An *averaged* notion of ICO quality underlies the *Area* quality metric of Sect. 3.1.B and [3]. The underlying quest is for schedules that maximize the *average* number of nodes that are eligible at each step of a DAG-execution.

Both the batched-ICO and Area quality measures admit optimal schedules for every DAG—but the general versions of both optimization problems are NP-Complete [17,20]. In the case of the Area measure, we were able to craft two readily computable associated heuristics (AO and SIDNEY) that are (empirically) computationally beneficial—as discussed at length in Sect. 3.2.B. Regrettably, we have not yet succeeded in finding such an associated heuristic for the batched version of the IC-scheduling paradigm. We leave the attractive challenges related to the batched paradigm to the interested reader.

B. Schedules with High AREA Quality. In contrast to the IC-scheduling paradigm, our major accomplishments with Area-oriented scheduling involved *heuristics* inspired by the paradigm. We begin our discussion with theoretical developments.

(1) *A-O schedulers for specific* DAG*-families.* In [3], we developed A-O schedulers for several classes of DAGs, including

- *monotonic tree*-DAG*s*: each DAG is either an *expansion-tree*—a DAG having one source, in which each nonsource has one parent—or the dual of an expansion-tree.
- *expansion-reduction* DAG*s*: each DAG is obtained by composing a k-sink expansion-tree with a k-source reduction-tree. (Imagine, e.g., that we match up the sources of the righthand tree in Fig. 1(b) with the sinks/leaves of the lefthand tree.)
- *compositions of bipartite cycle- and clique*-DAG*s*. (The "building-block" DAGs of Fig. 2(bottom) exemplify the cycles and cliques; the butterfly-DAG of Fig. 1(c) exemplifies the end product.)

Among the family-specific A-O schedulers that we developed, one stands out for its DAG-scheduling consequences. This is the efficient algorithm developed in [8], that produces A-O schedules for *series-parallel* DAG*s* (*SP*-DAGs, for short). (SP-DAGs have a rich history in the design of logic circuits. More recently, they have been used to model multi-threaded parallel computations; cf. [1].) This algorithm decomposes an input SP-DAG $\mathcal{G}$ according to the following recursive recipe for

generating SP-DAGs, and then it "reads off" an A-O schedule from the resulting "parse" of $\mathcal{G}$.

A *(2-terminal) series-parallel* DAG $\mathcal{G}$(*SP*-DAG) is produced by a sequence of the following operations.

1. Create. Form a DAG $\mathcal{G}$ that has:
   (a.) two nodes, a *source* s and a *target* t, which are jointly $\mathcal{G}$'s *terminals*,
   (b.) one arc, $(s \to t)$, directed from s to t.
2. Compose. SP-DAGs, $\mathcal{G}'$ with terminals s', t', and $\mathcal{G}''$, with terminals s'', t''.
   (a.) *Parallel composition.* Form the SP-DAG $\mathcal{G} = \mathcal{G}' \Uparrow \mathcal{G}''$ from $\mathcal{G}'$ and $\mathcal{G}''$ by merging s' with s'' to form a new source s and t' with t'' to form a new target t.
   (b.) *Series composition.* Form the SP-DAG $\mathcal{G} = (\mathcal{G}' \to \mathcal{G}'')$ from $\mathcal{G}'$ and $\mathcal{G}''$ by merging t' with s''. $\mathcal{G}$ has the single source s' and the single target t''.

(2) *The NP-completeness of AREA-maximization.* After we developed the ICO schedules for specific DAG-families discussed in Sect. 3.2.A(1), we were able to detect commonalities in reasoning that ultimately culminated in a proof for Theorem 2. In contrast, we found that our A-O schedules for specific DAG-families discussed in Sect. 3.2.B(1) relied in a fundamental way on the specific structures of the specific DAG-families. We were, therefore, not surprised to learn, in [20], that the general problem of computing A-O schedules is NP-complete. The proof in [20] reduces the 0–1 *Minimum Weighted-Completion-Time* Problem for bipartite DAGs, which is known to be NP-complete [25], to the A-O scheduling problem. This result shifted our focus entirely to the development of scheduling heuristics that (empirically) produced schedules with large Areas. We now describe the main heuristics that we have developed.

(3) *Area-oriented scheduling heuristics.* We have developed three scheduling heuristics that are "Area-centric," in the sense that they exploit Area-related structural properties of the DAG being scheduled.

(a) Heuristic D-G [3]. The *dynamic-greedy scheduling heuristic* D-G crafts a schedule for a DAG $\mathcal{G}$ by organizing $\mathcal{G}$'s eligible chores in a list structure that is (partially) ordered by chores' *yields,* with ties broken randomly. The *yield* $v(t)$ of eligible chore v at step t is the number of non-eligible chores that would be rendered eligible if v were executed now. The yield of a chore u can change at each step, and the execution of u can change the yields of many other chores, specifically, those that share children with u. Thus, in contrast with our other schedulers, the schedules produced by D-G change at each step—which gives D-G time-complexity commensurate with our other heuristics.

Note. D-G's successive choices of the next node to execute are *locally optimal*—except for its (nonexistent) tie-breaking mechanism.

(b) Heuristic AO [8]. The *Area-oriented scheduling heuristic* AO builds on two facts: (i) We have access to an efficient A-O scheduler for SP-DAGs; cf. Section 3.2.B(1). (ii) Every DAG $\mathcal{G}$ can be transformed efficiently to an SP-DAG $\sigma(\mathcal{G})$ that retains both $\mathcal{G}$'s inter-chore dependencies and (roughly) its degree

of inherent parallelism. Several sources describe "SP-izing" transformations; a perspicuous version from [12] is invoked in [8]. Heuristic AO produces a high-Area schedule for a DAG $\mathcal{G}$ in three steps.

Step 1. Transform $\mathcal{G}$ to an SP-DAG $\sigma(\mathcal{G})$, using an algorithm from [12].

Step 2. Produce an A-O schedule $\widetilde{\Sigma}$ for $\sigma(\mathcal{G})$, via the algorithm in [8].

Step 3. "Filter" schedule $\widetilde{\Sigma}$ to remove the "auxiliary" nodes added when SP-izing $\mathcal{G}$.

(c) Heuristic SIDNEY. The SIDNEY scheduling heuristic of [20] inherits both its name and its algorithmic underpinnings from a sophisticated DAG-decomposition scheme from [23]. It schedules an input DAG $\mathcal{G}$ in four steps.

Step 1. Transform $\mathcal{G}$ to its associated 0–1 *version* $\mathcal{G}_{0,1}$.

> The nodes of $\mathcal{G}_{0,1}$ are obtained by splitting each node v of $\mathcal{G}$ into two nodes, v_0 and v_1. Give each node of $\mathcal{G}_{0,1}$ that has a 0 subscript (the 0 nodes) a *processing time* of 0 and a *weight* of 1; give each node of $\mathcal{G}_{0,1}$ that has a 1 subscript (the 1-nodes) a processing time of 1 and a weight of 0. Finally, give $\mathcal{G}_{0,1}$ an arc $(u_1 \to v_0)$ for each arc $(u \to v)$ of $\mathcal{G}$ and an arc $(u_0 \to u_1)$ for each node u of $\mathcal{G}$.

Step 2. Use a max-flow computation to perform a Sidney decomposition of $\mathcal{G}_{0,1}$, via the algorithm in [23].

Step 3. Say that the decomposition of $\mathcal{G}_{0,1}$ produces DAGs $\mathcal{G}_1, \ldots, \mathcal{G}_k$.

a. Remove all 0-nodes from every $\mathcal{G}_i$.

b. Use heuristic D-G to produce a schedule Σ_i for each $\mathcal{G}_i$.

Step 4. Output schedule $\Sigma \stackrel{\text{def}}{=} \Sigma_1 \cdots \Sigma_k$, the concatenation of the k subschedules.

At the cost of somewhat more computation than needed for heuristics D-G and AO, SIDNEY empirically produces schedules whose Areas are within 85 % of maximal [20].

3.3 The Benefits of Opportunistic Scheduling

A. Benefits Exposed via Simulation Experiments. Simulation-based studies of the opportunistic scheduling paradigms we have discussed appear in [3,4,13,16,24]. Rather than reproduce material that appears in great detail in those sources, I have decided to summarize here the major *messages* of those studies.

B1. One observes in all of the cited sources that there are two circumstances under which all (oblivious) scheduling paradigms are essentially equivalent in performance.

(a) When computing resources are plentiful, then the inherently sequential critical path of a DAG is the only constraint on the speed of executing the DAG.

(*b*) When computing resources are really meager, then there are no opportunities for efficiency-enhancing concurrency.

B2. One observes in [13,16] situations where ICO schedules outperform a variety of platform-oblivious competitor-schedules by as much a 10–20%. The tested workloads in [16] were real scientific computations; the ones in [13] were synthetic, but with structures that approximated those of real scientific computations.

B3. Since the Area quality-metric is a weakening of IC-quality, it is not surprising that the benefits of A-O schedules observed in [3] are more modest than those observed in the IC-frameworks of [13,16]. That said, one still often observes A-O schedules outperforming a variety of platform-oblivious competitor-schedules by double-digit percentages. Indeed, the same type of performance is observed even in [4] with heuristic AO. One observes that A-O schedules outperform those produced by heuristic AO, but only by percentages that may not justify the computational cost of producing the A-O schedule.

B4. The experiments in [4] suggest that the schedules produced by heuristic AO perform best when computing resources become available according to distributions that have low variances.

B5. The experimental settings in [3,4] (involving, respectively, A-O schedules and schedules produced by heuristic AO) posit that computing resources become available according to low-variance distributions. It is observed experimentally in [4] that, within such settings, the Areas of generated schedules inversely track the makespans of the schedules' DAG-executions—i.e., larger Areas correlate with smaller makespans.

B6. In contrast to heuristic AO, the very high-Area schedules produced by heuristic SIDNEY seem to favor situations wherein the distributions governing computing-resource availability do not have low variances [20]. This suggests that SIDNEY's schedules may be desirable in settings such as enterprise clouds, where the user can tailor the purchase of computing resources based on the varying numbers of eligible nodes produced over time by one's DAG-schedule.

B7. The experiments reported in [24] seem to validate Observations 4 and 6: schedules produced by heuristic AO are observed to perform very well in "single-instance" enterprise clouds, wherein there is a single block of computing resources that are available at any moment. In fact, the static heuristic AO i sobserved to compete well with dynamic competitor schedules.

B. Two Major Open Issues. We close with a two open issues regarding opportunistic DAG-scheduling. The benefits we have already uncovered—and enumerated in this section—explain our belief in the potential significance of success in addressing these issues.

Q1. The discovery in [19] that many DAGs do not admit ICO schedules led to three weakened version of IC-scheduling: batched IC-scheduling [18], a version

based on weakening the $\triangleright$-priority relation of Theorem 2 [16], and Area-oriented scheduling [3]. Of these alternatives, only Area-oriented scheduling has been studied in any detail. The other alternatives certainly deserve more attention than they have received.

Q2. Our study of opportunistic DAG-scheduling began with a focus on *dynamically heterogeneous* computing platforms—and it has largely retained that focus. The benefits of eligible-node-enhancing DAG-schedules should be significant also in other domains:

(*a*) Opportunistic DAG-schedulers may be valuable when pursuing cost-effective computing within an enterprise cloud. Having access to large numbers of eligible nodes should alow a user to maximally exploit available cost-effective resources. This benefit is hinted at in [24], but it deserves careful study.

(*b*) In a similar vein, opportunistic DAG-schedulers may be beneficial in power-aware computing environments. Their schedules may enable one to maximally exploit low-power resources as they become available. This possibility, too, deserves careful study.

Acknowledgments. It is a pleasure to acknowledge the invaluable contributions of my collaborators on the work discussed here: Gennaro Cordasco, Rosario De Chiara, Ian Foster, Robert Hall, Greg Malewicz, Rajmohan Rajaraman, Scott Roche, Mark Sims, Michela Taufer, Arun Venkataramani, Mike Wilde, Matt Yurkewych. Our work on opportunistic DAG-scheduling has been supported in part by several grants from the US National Science Foundation, most recently Grant CSR-1217981.

References

1. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: an efficient multithreaded runtime system. In: 5th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming (PPoPP 1995) (1995)
2. Casanova, H., Dufossé, F., Robert, Y., Vivien, F.: Scheduling parallel iterative applications on volatile resources. In: 25th IEEE International Parallel and Distributed Processing Symposium (2011)
3. Cordasco, G., De Chiara, R., Rosenberg, A.L.: On scheduling DAGs for volatile computing platforms: area-maximizing schedules. J. Parallel Distrib. Comput. **72**(10), 1347–1360 (2012)
4. Cordasco, G., De Chiara, R., Rosenberg, A.L.: An AREA-oriented heuristic for scheduling DAGs on volatile computing platforms. IEEE Trans. Parallel Distrib. Syst. **26**(8), 2164–2177 (2015)
5. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Applying IC-scheduling theory to some familiar classes of computations. In: Workshop on Large-Scale, Volatile Desktop Grids (PCGrid 2007) (2007)
6. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Advances in IC-scheduling theory: scheduling expansive and reductive DAGs and scheduling DAGs via duality. IEEE Trans. Parallel Distrib. Syst. **18**, 1607–1617 (2007)
7. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Extending IC-scheduling via the Sweep algorithm. J. Parallel Distrib. Comput. **70**, 201–211 (2010)

8. Cordasco, G., Rosenberg, A.L.: On scheduling series-parallel DAGs to maximize AREA. Int. J. Found. Comput. Sci. **25**(5), 597–621 (2014)
9. Cordasco, G., Rosenberg, A.L., Sims, M.: On clustering DAGs for task-hungry computing platforms. Cent. Eur. J. Comput. Sci. **1**, 19–35 (2011)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (1999)
11. Estrada, T., Taufer, M., Reed, K.: Modeling job lifespan delays in volunteer computing projects. In: 9th IEEE International Symposium on Cluster, Cloud, and Grid Computing (CCGrid) (2009)
12. González-Escribano, A., van Gemund, A.J.C., Cardeñoso-Payo, V.: Mapping unstructured applications into nested parallelism. In: Palma, J.M.L.M., Sousa, A.A., Dongarra, J., Hernández, V. (eds.) VECPAR 2002. LNCS, vol. 2565, pp. 407–420. Springer, Heidelberg (2003)
13. Hall, R., Rosenberg, A.L., Venkataramani, A.: A comparison of DAG-scheduling strategies for internet-based computing. In: 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS) (2007)
14. Kale, L.V., Bhatele, A. (eds.): Parallel Science and Engineering Applications: The Charm++ Approach. New York, Taylor & Francis Group, CRC Press (2013)
15. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Lebofsky, M.: SETI@home: massively distributed computing for SETI. In: Dubois, P.F (ed.) Computing in Science and Engineering. IEEE Computer Society Press (2000)
16. Malewicz, G., Foster, I., Rosenberg, A.L., Wilde, M.: A tool for prioritizing DAGMan jobs and its evaluation. J. Grid Comput. **5**, 197–212 (2007)
17. Malewicz, G., Rosenberg, A.L.: Batch-scheduling dags for internet-based computing. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 262–271. Springer, Heidelberg (2005)
18. Malewicz, G., Rosenberg, A.L.: A pebble game for internet-based computing. In: Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.) Theoretical Computer Science. LNCS, vol. 3895, pp. 291–312. Springer, Heidelberg (2006)
19. Malewicz, G., Rosenberg, A.L., Yurkewych, M.: Toward a theory for scheduling DAGs in internet-based computing. IEEE Trans. Comput. **55**, 757–768 (2006)
20. Roche, S.T., Rosenberg, A.L., Rajaraman, R.: On constructing DAG-schedules with large AREAs. Concurrency Comput. Pract. Experience **27**(16), 4107–4121 (2015)
21. Rosenberg, A.L.: On scheduling mesh-structured computations for internet-based computing. IEEE Trans. Comput. **53**, 1176–1186 (2004)
22. Rosenberg, A.L., Yurkewych, M.: Guidelines for scheduling some common computation-DAGs for internet-based computing. IEEE Trans. Comput. **54**, 428–438 (2005)
23. Sidney, J.B.: Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. Oper. Res. **23**(2), 283–298 (1975)
24. Taufer, M., Rosenberg, A.L.: Scheduling DAG-based workflows on single cloud instances: high performance and cost effectiveness with a static scheduler. Int. J. High Perform. Comput. Appl. (2015). doi:10.1177/1094342015594518
25. Woeginger, G.J.: On the approximability of average completion time scheduling under precedence constraints. Discrete Appl. Math. **131**(1), 237–252 (2003)
26. Yao, S., Lee, H.-H.S.: Using mathematical modeling in provisioning a heterogeneous cloud computing environment. IEEE Comput. **44**, 55–62 (2011)
27. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving MapReduce performance in heterogeneous environments. In: 7th USENIX Symposium on Operating System Design and Implementation (2008)

http://www.springer.com/978-3-319-43658-6

Euro-Par 2016: Parallel Processing
22nd International Conference on Parallel and Distributed Computing, Grenoble, France, August 24-26, 2016, Proceedings
Dutot, P.-F.; Trystram, D. (Eds.)
2016, XXIX, 699 p. 236 illus., Softcover
ISBN: 978-3-319-43658-6