A Model for Cycle-Stealing in Networks of Workstations: Progress and Problems

by
A. L. Rosenberg

CMPSCI Technical Report 98-73
July, 1998

A Model for Cycle-Stealing in Networks of Workstations: Progress and Problems*

Arnold L. Rosenberg
Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
rsnbrg@cs.umass.edu

July 9, 1998

Abstract

We present a formal model for data-parallel cycle-stealing in networks of workstations that was developed in

> S.N. Bhatt, F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg. On optimal strategies for cycle-stealing in networks of workstations. *IEEE Trans. Comp.*, 46:545–557, 1997

and refined in

> A.L. Rosenberg. Guidelines for data-parallel cycle-stealing in networks of workstations, I: on maximizing expected output. Tech. Rpt. 98-15, Univ. Massachusetts. See also *12th IEEE Intl. Parallel Proc. Symp.*, 519–523, 1998.
> A.L. Rosenberg. Guidelines for data-parallel cycle-stealing in networks of workstations, II: on maximizing guaranteed output. In preparation, Univ. Massachusetts, 1998.

We review the major results about scheduling cycle-stealing opportunities that have been obtained in those sources; and we enumerate and discuss several significant problems that remain unresolved.

1 Introduction

The cycle-stealing problem. Cycle-stealing is an emerging modality of parallel computation that occurs within a network of workstations (NOW). In the version of cycle-stealing that we

*The research reported herein was supported in part by NSF Grant #CCR-97-10367. This paper encapsulates an invited address at the Intl. Conf. on FUN with Algorithms, Isle of Elba, June 18-20, 1998.

study, the owner of workstation A contracts to take control of workstation B's processor when B's owner is away, with the promise of relinquishing control of B *immediately* upon the owner's return—thereby losing all work in progress.[1] Such a contract presents a challenging scheduling dilemma for A's owner. On the one hand, the typically large overhead required to set up an inter-workstation communication in a NOW suggests that A's owner should communicate with B very infrequently, sending large quantities of work in each transmission—in order to minimize the aggregate nonproductive communication setup time. On the other hand, the harsh interrupt provision of the contract suggests that A's owner should communicate with B very frequently, sending small quantities of work in each transmission—in order to keep the amount of vulnerable remote work small at all times. Avenues for resolving this dilemma form the focus of our study.

Related work. The conceptual and economic rationale for using NOWs as parallel computers is discussed at length in [10]. Several anecdotal descriptions of such use appear in the applications literature; cf. [16]. Systems too numerous to list have been developed for facilitating the mechanics of cycle-stealing in NOWs, e.g., providing a mechanism that allows A to assume control of B's processor; cf. [8, 11]. The scheduling problem for cycle-stealing opportunities has thus far received relatively little attention. In [1], one orchestrates a cycle-stealing opportunity by "auctioning off" large identical chunks of a compute-intensive task within a NOW, to determine the sub-NOW that promises the best parallel speedup (computed using the authors' model); one then distributes appropriate-size chunks of the task within the "winning" sub-NOW. In [2], cycle-stealing is viewed as one application, among many, of a theory of how to make random decisions better than by random choices; with high probability, the proposed randomized scheduling algorithm achieves work-output that is within a logarithmic factor of optimal. The companion papers [4, 5] present and analyze a system that schedules dags (directed acyclic graphs) on a NOW in a way that optimizes system time and space requirements. One finds in [14, 15] two opposing systems-oriented philosophies for scheduling cycle-stealing ("pushing" and "pulling"), but neither formal models nor rigorous analyses. Finally, there are our three studies of cycle-stealing, which form the subject matter of this paper. In [3], we developed a two-faceted formal model of *data-parallel* cycle-stealing (wherein the computational load consists of an enormous number of small independent tasks); one submodel focuses on the *expected* work-output of a cycle-stealing opportunity, while the other focuses on the *guaranteed* work-output. One finds in [3] exactly optimal strategies for a variety of specific scenarios within both submodels. In [12, 13], we refine the model of [3], and we derive guidelines for near-optimal data-parallel cycle-stealing for a broad range of scenarios within the two submodels of [3]: [12] focuses on the expected-output submodel and [13] on the guaranteed-output submodel.

The current paper reviews the models and main results of [3, 12, 13] (Sections 2 and 3) and then presents and discusses a number of significant unresolved problems that merit further study (Section 4).

[1] Such a draconian contract is inevitable, for instance, when workstation B is a laptop that can be unplugged from the network; it occurs also, we are told, in cycle-stealing contracts in force at many institutions.

2 A Formal Model for Data-Parallel Cycle-Stealing

2.1 General Features of the Model

The computing environment. We assume that the workstations in the NOWs of interest are identical in structure and power. Cleaving to the "architecture-independent" scheduling paradigm of [9], we characterize each NOW by a single parameter, the fixed constant c which is the overhead for setting up each pair of communications in which workstation A sends work to workstation B and B returns the results of that work to A. Note that c is independent of the marginal per-task cost of communications between A and B, which we incorporate into the cost of computing the tasks. We assume that tasks are indivisible; they may differ in size, but we know their sizes perfectly.

Cycle-stealing opportunities and schedules. We view each cycle-stealing opportunity as a sequence of *episodes* during which workstation A has access to workstation B, punctuated by *interrupts* caused by the return of B's owner. In order to decrease our vulnerability to the metaphorical malicious adversary's ability to kill work in progress on B, we partition each episode into *periods*, each of which begins with A sending work to B and ends with B returning the results of that work. Since our discretionary power thus resides solely in deciding how much work to send in each period, we view an *episode-schedule* simply as a sequence of period-lengths $\mathcal{S} = t_1, t_2, \ldots$. As we shall see, we usually employ a different episode-schedule for each episode in a cycle-stealing opportunity.

Work-output. Reflecting the inescapability of the communication setup time c, a length-t period in an episode accomplishes[2] $t \ominus c$ units of work if the period is not interrupted and 0 units of work if the period is interrupted. Thus, the entire episode scheduled by $\mathcal{S} = t_1, t_2, \ldots$ accomplishes

$$\mathcal{W}(\mathcal{S}) \;=\; \sum_{i=1}^{k-1} (t_i \ominus c)$$

units of work when it is interrupted during period k. The work garnered in an entire cycle-stealing opportunity is the sum of the work-outputs of the opportunity's constituent episodes.

Approaching our submodels. In order to have any chance of achieving productive work during a cycle-stealing opportunity, we must have access to some mechanism that prevents a malicious adversary from interrupting the first period of every episode, thereby killing all work that we start. Within our model, we consider two such mechanisms which, while idealized, are not completely unrealistic in any actual environment where we may have some knowledge of B's owner's work habits (say, via traces). These mechanisms, which define our two submodels, are the subjects of the next two subsections.

[2] "$x \ominus y$" denotes *positive subtraction*: $x \ominus y = x - y$ if $x \geq y$ and 0 otherwise.

2.2 The Known-Risk (KR) Submodel

The KR submodel, which focuses on scheduling a single episode of cycle-stealing, assumes that we know the probability distribution of interrupts by B's owner—inferred possibly from statistics concerning the lengths of workstation B's idle periods. This knowledge is encapsulated in the *life function* of the episode, i.e., the function

$$\mathcal{P}(t) \stackrel{\text{def}}{=} Pr(\text{being "alive" at time } t).$$

Our challenge within the KR submodel is to maximize the *expected* work production from the episode characterized by $\mathcal{P}$; for any episode-schedule $\mathcal{S}$:

$$\text{Exp-Work}(\mathcal{S}) = \sum_{i \geq 1} (t_i \ominus c)\mathcal{P}(T_i) \tag{1}$$

where, for each i, $T_i \stackrel{\text{def}}{=} t_1 + t_2 + \cdots + t_i$.

Note: The simple form of Exp-Work($\mathcal{S}$) makes it easy to cope with life functions that are known *approximately*—say via gathered statistics.

2.3 The Guaranteed-Output (GO) Submodel

The GO submodel, which focuses on scheduling a multi-episode cycle-stealing opportunity, assumes that we have modified the cycle-stealing contract so that:

- workstation A gets a guaranteed amount of usable time on B, say a *usable lifespan* of U time units;
- the adversary gets only a fixed allocation of interrupts, say p interrupts, during the lifespan.

Our challenge within the GO submodel is to maximize the *guaranteed* work-output from the entire cycle-stealing opportunity, no matter how many interrupts actually occur, or where they occur.

2.4 An Important Auxiliary Result

The following technical result is proved for the KR submodel in [3], with a slight strengthening in [12]; it is proved for the GO submodel in [13]. The result has three significant consequences. First, it materially narrows our search for optimal schedules. Second, it allows us to use ordinary subtraction, rather than positive subtraction, when discussing the potential work-output from each of an episode's periods, save the last. Finally, within the GO submodel, it leads directly to two significant observations about the adversary's strategy during the game. The result shows that we lose no generality by restricting attention to episode-schedules that are *productive*, in the sense of having all period-lengths, save perhaps the last, exceed c. Stated formally,

The Productivity Theorem [3, 12, 13] *Any episode-schedule $\mathcal{S}$ can be replaced by an episode-schedule $\mathcal{S}'$ whose work-output is at least as great, such that each period of $\mathcal{S}'$, save perhaps the last, has length $> c$.*

In fact, in studying the GO submodel in [13], we have focused on episode-schedules that are *fully productive*, in the sense that *all* period-lengths—including the last—exceed c. We have not yet been able to verify that this focus loses no generality, but it seems to make sense intuitively.

3 Progress To Date

For the KR submodel, the main results of [12] specify each period-length t_k of schedule $\mathcal{S} = t_1, t_2, \ldots, t_m$ from all *lower*-index period-lengths. For the GO submodel, the main results of [13] specify each period-length t_k from all *higher*-index period-lengths. In both cases, however, our results specify the base period-length—t_1 for the KR submodel and t_m for the GO submodel—only to within a factor of 2 (and possibly with side conditions).

3.1 Progress within the KR Submodel

The results reported in this subsection come from [12] unless otherwise attributed.

3.1.1 Guidelines for Scheduling within the KR Submodel

Let schedule $\mathcal{S} = t_1, t_2, \ldots$ be optimal for the *differentiable* life function $\mathcal{P}$.

A. Characterizing the optimal sequence of t_i's

For each period-index $k \geq 1$:[3]

$$\mathcal{P}(T_k) \;=\; -\sum_{j \geq k} (t_j - c)\mathcal{P}'(T_j).$$

In computationally more useful form: for each period-index $k > 1$:

$$\mathcal{P}(T_k) \;=\; \mathcal{P}(T_{k-1}) + (t_{k-1} - c)\mathcal{P}'(T_{k-1}).$$

Note that the preceding specification can be invoked in an on-line fashion, determining t_{i+1} only after the ith period. In such a setting, one can use *conditional*, rather than *a priori*, probabilities in defining an episode's life function.

B. Bounds on the optimal t_1

[3] As usual, "f'" denotes the first derivative of the differentiable function f.

For convex[4] *life functions* $\mathcal{P}$:

$$\sqrt{\frac{c^2}{4} - \frac{c\mathcal{P}(t_1)}{\mathcal{P}'(t_1)}} + \frac{c}{2} \;<\; t_1 \;\leq\; 2\sqrt{\frac{c^2}{4} - \frac{c\mathcal{P}(t_1)}{\mathcal{P}'(t_1)}} + c.$$

For concave *life functions* $\mathcal{P}$:

$$\sqrt{\frac{c^2}{4} - \frac{c\mathcal{P}(t_1)}{\mathcal{P}'(t_1)}} + \frac{c}{2} \;<\; t_1 \;\leq\; 2\sqrt{\frac{c^2}{4} - \frac{c\mathcal{P}(t_1)}{\mathcal{P}'(t_1/2)}} + c.$$

3.1.2 Applications of the KR Guidelines

In this section, we compare the period-specifications obtained using the broadly applicable guidelines of [12] with the *ad hoc* optimal specifications derived in [3] for the KR submodel (the "BCLR values"). Throughout, we denote the sought schedule by $\mathcal{S} = t_1, t_2, \ldots$.[5]

A. The Geometrically Decreasing Lifespan (GDL) Scenario

In the GDL scenario, each episode has a "half-life." The scenario's life function is $\mathcal{P}_a(t) \stackrel{\text{def}}{=} a\mathcal{P}_a(t+1) = a^{-t}$.

The optimal GDL schedule	
The sequence of t_i's	
Our predicted values	$a^{-t_i} + t_{i-1}\ln a \;=\; 1 + c\ln a$
The optimal BCLR values	$a^{-t_i} + t_i \ln a \equiv 1 + c\ln a$
The value of t_1	
Our predicted range	$\sqrt{\frac{c^2}{4} + \frac{c}{\ln a}} + \frac{c}{2} \;\leq\; t_1 \;\leq\; c + \frac{1}{\ln a}$
The optimal BCLR value	$t_1 + \frac{a^{-t_1}}{\ln a} \;=\; c + \frac{1}{\ln a}$

[4] A function f is *convex* (resp., *concave*) if its derivative is everywhere nondecreasing (resp., everywhere nonincreasing): for all positive real ξ and $\eta > \xi$, we have $f'(\xi) \leq f'(\eta)$ (resp., $f'(\xi) \geq f'(\eta)$).

[5] We denote by "$\ln x$" (resp., "$\log x$") the base-e (resp., the base-2) logarithm.

Ad hoc hint. The BCLR values derive from the fact that the conditional risk of interruption is identical after every period in the GDL scenario.

B. The Uniform Risk (UR) Scenario

In the UR scenario, the risk of interruption is uniform throughout an L-unit *episode-lifespan*. The scenario's life function is $\mathcal{P}_L(t) \stackrel{\text{def}}{=} 1 - t/L$.

The optimal UR schedule	
The sequence of t_i's	
Our predicted values	$t_i = t_{i-1} - c$
The optimal BCLR values:	$t_i = t_{i-1} - c$
The value of t_1	
Our predicted range	$\sqrt{cL} \leq t_1 \leq 2\sqrt{cL} + 1$
The optimal BCLR value	$t_1 = \sqrt{2cL} +$ l.o.t.

Ad hoc hint. The BCLR values derive from the fact that the aggregate communication overhead forms an arithmetic sum.

C. The Geometrically Increasing Risk (GIR) Scenario

In the GIR scenario, the risk of interruption doubles at every step of the L-unit episode-lifespan. The scenario's life function is $\mathcal{P}^g_L(t) \stackrel{\text{def}}{=} (2^L - 2^t)/(2^L - 1)$.

The optimal GIR schedule	
The sequence of t_i's	
Our predicted values	$t_{k+1} = \log((t_k - c)\ln 2 + 1)$
The optimal BCLR values	$t_{k+1} = \log(t_k - c + 2)$
The value of t_1	
Our predicted value	$t_1 = \dfrac{L}{\log^2 L} + \text{l.o.t.}$
The optimal BCLR value	Given implicitly

Ad hoc hint. The BCLR values emerge from explicitly comparing the optimal episode-schedule $\mathcal{S}$ with its *kth-period perturbations*:

$$\mathcal{S}^{(\pm k)} \stackrel{\text{def}}{=} t_1, t_2, \ldots, t_{k-1}, t_k \pm 1, t_{k+1} \mp 1, t_{k+2}, \ldots, t_{m-1}.$$

Our guideline-generated values implicitly use the same device.

3.2 Progress within the GO Submodel

The results reported in this subsection come from [13].

3.2.1 Guidelines for Scheduling within the GO Submodel

Our quest for good schedules within the GO submodel employs the following strategy. For each combination of lifespan U and number p of potential interrupts, we craft the doubly parameterized episode-schedule[6]

$$\mathcal{S}^{(p)}[U] = t_1^{(p)}[U],\ t_2^{(p)}[U], \ldots,\ t_m^{(p)}[U],$$

where $m = m^{(p)}[U]$. At the beginning of each lifespan-U p-interrupt opportunity, workstation A invokes episode-schedule $\mathcal{S}^{(p)}[U]$. If the opportunity is interrupted at time T, then upon regaining control of B's processor, A invokes episode-schedule $\mathcal{S}^{(p-1)}[U - T]$. A proceeds in this way until the entire usable lifespan is exhausted.

[6] To enhance legibility, we omit the parameters p and/or U when they are clear from context.

Throughout, let $\mathcal{W}^{(p)}[U]$ denote the maximum achievable guaranteed work-output from a lifespan-U p-interrupt opportunity.

A. The Boundary Situations

Short lifespans: *If* $U \leq (p+1)c$, *then* $\mathcal{W}^{(p)}[U] = 0$.

The "end game": *When* $p = 0$, *the* 1*-period schedule* $\mathcal{S}^{(0)}[U] = U$ *achieves* $\mathcal{W}^{(0)}[U] = U - c$ *units of work, which is optimal.*

B. The Optimal Period-Lengths

The optimal episode-schedule $\mathcal{S}^{(p)}[U] = t_1^{(p)}[U],\ t_2^{(p)}[U], \ldots,\ t_m^{(p)}[U]$ *satisfies the following.*

Case 1. $p = 0$**:** $m^{(0)}[U] = 1$, *and* $t_1^{(0)}[U] = U$.

Case 2. $p > 0$**:** *Let* ℓ *be the smallest period-index for which*

$$U - T_{\ell-1}^{(p)} \ = \ t_\ell^{(p)} + t_{\ell+1}^{(p)} + \cdots + t_m^{(p)} \ > \ pc.$$

Then the following table (partially) specifies the optimal period-lengths of $\mathcal{S}^{(p)}[U]$.[7]

Range of k	Value/Range of $t_k^{(p)}$
$1 \leq k \leq \ell - 2$	$t_k^{(p)} \ = \ c + \mathcal{W}^{(p-1)}[U - T_k^{(p)}] - \mathcal{W}^{(p-1)}[U - T_{k+1}^{(p)}]$
$k = \ell - 1$	$t_k^{(p)} \ = \ c + \mathcal{W}^{(p-1)}[t_\ell^{(p)}]$
$\ell \leq k \leq m$	$t_k^{(p)} \ \in \ (c,\ 2c]$

3.2.2 Applications of the GO Guidelines

We illustrate in the following table the approximate optimal values of the scheduling parameters for the cases $p = 1$ and $p = 2$. We must satisfy ourselves with approximate values because of the computationally complicated expressions for the actual values—which often contain nested radicals and, even worse, are often not even algebraic. This issue is discussed at length in [13].

[7] As usual, "$x \in (c,\ 2c]$" (resp., "$x \in (c,\ 2c)$") means "$c < x \leq 2c$" (resp., "$c < x < 2c$").

Parameter	Approximate Optimal Value	
	$p = 1$	$p = 2$
$m^{(p)}$	$\sqrt{2U/c}$	$\gamma\sqrt{2U/c}$ $\gamma \in (1.54,\ 1.62)$
α	$c/2$	N/A
$\alpha + \beta$	N/A	$\in (1.912,\ 2]$
$t_k^{(p)}$ $k \leq m-3$	$\sqrt{2cU} - kc$	$t_{m-2}^{(2)} + \delta_k(m^{(2)} - k - 2)c$ $\delta_k \in (0.38,\ 0.42)$
$t_{m-2}^{(p)}$	$\sqrt{2cU} - (m-2)c$	$\left(5/2 + \alpha + \beta - \sqrt{4 + 2(\alpha+\beta)}\right) c$
$t_{m-1}^{(p)}$	$3c/2$	$c + \beta \in (c,\ 2c]$
$t_m^{(p)}$	$3c/2$	$c + \alpha \in (c,\ 2c]$
$\mathcal{W}^{(p)}$	$U - \sqrt{2cU} - c/2$	$U - (1+\epsilon)\sqrt{2cU} - c/2$ $\epsilon \in (0.62,\ 0.65)$

4 Remaining Challenges

4.1 Challenges within the Current Model

4.1.1 For Both Submodels

Seeking definitive schedule specifications. In our view, the most significant shortcoming in our progress thus far on both of our submodels is that our guidelines are not totally prescriptive—due to the factor-of-2 uncertainty in the "base" period-lengths (t_1 in the KR model and the highest-indexed period-lengths in the GO model). It is conceivable that some uncertainty in specifying the t_i is inevitable, for we do not yet know if optimal schedules are unique within either submodel—an interesting question in its own right. We do have two observations that are relevant to the uniqueness question. First, every specific scenario that we have looked at thus far admits a unique optimal schedule—if it admits any optimal schedule at all. Second, any nonuniqueness that occurs must reside in the factor-of-2 uncertainty just mentioned.

Proceeding with less exact knowledge. In their current forms, both of our submodels demand a lot of exact knowledge—exact task times and durations of lifespans in both submodels, the exact form of the life function in the KR submodel, the exact number of potential interrupts in the GO submodel. As noted earlier, the simple form of the expected-work summation (1) for the KR submodel should allow us to estimate the impact of using a life function that only approximates the true probability of "being alive", say one obtained by fitting a smooth curve to trace data. Similarly relaxing any of the other demands for exact knowledge seems to be a much stiffer challenge—but a worthwhile one since the range of potential applications of our model to real computational situations would be significantly enhanced if such relaxation were

possible. One possible avenue for relaxation that was suggested by Kurt Mehlhorn would be to replace some or all of our assumed exact knowledge by the assumption that we know (bounds on) the *expected* lengths of episodes.

4.1.2 For the GO submodel

Understanding GO schedules. The earlier mentioned computational complexity of the submodel has prevented us to this point from explicitly determining an/the optimal schedule for a GO scenario with $p > 2$ possible interrupts—even though such determination is clearly possible in principle. A major reason for wanting to proceed to larger values of p is to be able to verify or refute the following conjecture.

Conjecture. *There is a fixed constant ξ such that, for all p and U, $\mathcal{W}^{(p)}[U] > U - \xi\sqrt{2cU} - c$.*

We are currently pursuing this conjecture in [13].

Fully productive vs. productive schedules. Although of limited likely import, the following vexing problem merits some attention. Can every episode-schedule be replaced by a fully productive one without compromising guaranteed work-output? An affirmative answer seems likely, since having a nonproductive last period (the only one that need not be productive, as we saw in the Productivity Theorem in Section 2.4) seems only to help the adversary by nullifying part of our usable lifespan. But, a rigorous answer has thus far eluded us.

4.1.3 For the KR Submodel

Understanding the greedy heuristic. There is a simple on-line scheduling heuristic for this submodel, which would seem intuitively to yield good work-output, at least for some life functions. This is the "greedy" heuristic which operates as follows.

1. Choose the initial period-length t_1 as the value of t that maximizes the function $\mathcal{P}_1(t) \stackrel{\text{def}}{=} (t-c)\mathcal{P}(t)$. ($\mathcal{P}_1$ is just the first term of the expected-work summation (1) for a productive episode-schedule.)

2. Having thus chosen t_1, choose the second period-length t_2 as the value of t that maximizes the function $\mathcal{P}_2(t) \stackrel{\text{def}}{=} (t-c)\mathcal{P}(t+t_1)$.

3. Having thus chosen t_1 and t_2, choose the third period-length t_3 as the value of t that maximizes the function $\mathcal{P}_3(t) \stackrel{\text{def}}{=} (t-c)\mathcal{P}(t+t_1+t_2)$.

4. And so on...

We prove in [12] that this strategy never generates the *optimal* sequence of period-lengths, but we have not shown that it does not yield a computationally simple *good* approximation to

optimal work-output, at least for some large class of life functions. This is worth looking into, for possible practical scheduling.

Life functions that (do not) admit optimal schedules. Of somewhat less moment than the other questions in this section, but intriguing nonetheless, is the fact that, within the KR submodel, some life functions—$\mathcal{P}(t) = 1/(t+1)$ is one example—do not admit an optimal schedule [12]. We have yet to understand: (a) why this is so (except in a purely mathematical sense); (b) how to characterize those life functions that do admit optimal schedules; (c) how to cope with a cycle-stealing opportunity whose life function is intransigent in this way.

4.2 Significant Alternative Submodels

There are (at least) two major directions in which our model should be enhanced.

Less draconian contracts. Perhaps the biggest criticism we have heard concerning our model is its insistence that interrupts kill work in progress. Despite our rejoinders about laptops being unplugged and about the several instantiations of our type of harsh contract actually in force, many people have indicated that they would prefer a modification to our model wherein interrupted jobs get "niced"—i.e., get lower priority—rather than being killed. We believe that both types of contracts merit study, so we in fact hope that a "niced" variant of our model gets some of the attention it deserves. We remark that such a "niced" variant of our model would assume some of the characteristics of scheduling within heterogeneous clusters of computers [6], just as our present variant enjoys some of the characteristics of scheduling in anticipation of machine failures [7].

More complicated workloads. It would be desirable to extend the scheduling approach that gives rise to our results in Section 3 to handle tasks that are not mutually independent. A natural first step in this direction would be to try to schedule computations that have the structure of dags. Such scheduling is, of course, a well-studied problem in other parallel computing environments [6, 9] but has only begun to be studied within the context of computing in clusters [4, 5].

References

[1] M.J. Atallah, C.L. Black, D.C. Marinescu, H.J. Siegel, and T.L. Casavant. Models and algorithms for coscheduling compute-intensive tasks on a network of workstations. *J. Parallel Distr. Comput.*, 16:319–327, 1992.

[2] B. Awerbuch, Y. Azar, A. Fiat, and F.T. Leighton. Making commitments in the face of uncertainty: how to pick a winner almost every time. *28th ACM Symp. on Theory of Computing*, 519–530, 1996.

[3] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg. On optimal strategies for cycle-stealing in networks of workstations. *IEEE Trans. Comp.*, 46:545–557, 1997.

[4] R. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. *35th IEEE Symp. on Foundations of Computer Science*, 356–368, 1994.

[5] R. Blumofe and D.S. Park. Scheduling large-scale parallel computations on networks of workstations. *3rd Intl. Symp. on High-Performance Distributed Computing*, 96–105, 1994.

[6] S. Chen, M.M. Eshaghian, and Y.-C. Wu. Mapping arbitrary non-uniform task graphs onto arbitrary non-uniform system graphs. *Intl. Conf. on Parallel Processing*, II:191–194, 1995.

[7] E.G. Coffman, Jr., L. Flatto, and A.Y. Krenin. Scheduling saves in fault-tolerant computations. *Acta. Inform.*, 30:409–423, 1993.

[8] M. Mutka and M. Livny. Scheduling remote processing capacity in a workstation-processor bank network. *7th Intl. Conf. on Distr. Computing Systs.*, 2–9, 1987.

[9] C.H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms. *SIAM J. Comput.*, 19:322–328, 1990.

[10] G.F. Pfister. *In Search of Clusters.* Prentice-Hall, Upper Saddle River, N. J., 1995.

[11] A. Reinefeld, J. Gehring, and M. Brune. Communication across parallel message-passing environments. *J. Systs. Architecture on Cluster Computing*, 1997.

[12] A.L. Rosenberg. Guidelines for data-parallel cycle-stealing in networks of workstations, I: on maximizing expected output. Tech. Rpt. 98-15, Univ. Massachusetts. See also *12th IEEE Intl. Parallel Proc. Symp.*, 519–523, 1998.

[13] A.L. Rosenberg. Guidelines for data-parallel cycle-stealing in networks of workstations, II: on maximizing guaranteed output. In preparation, Univ. Massachusetts, 1998.

[14] M. Stumm. The design and implementation of a decentralized scheduling facility for a workstation cluster. *2nd IEEE Conf. on Computer Workstations*, 12–22, 1988.

[15] M.M. Theimer and K.A. Lantz. Finding idle machines in a workstation-based distributed environment. *IEEE Trans. Software Eng'g.*, 15:1444–1458, 1989.

[16] S.W. White and D.C. Torney. Use of a workstation cluster for the physical mapping of chromosomes. *SIAM NEWS*, March, 1993, 14–17.