Proceedings of the 28th Annual Hawaii International Conference on System Sciences — 1995

Compile-Time Scheduling with Resource-Constraints.

Greet Bilsen, Rudy Lauwereins, J.A. Peperstraete

ESAT-ACCA-Laboratory, Katholieke Universiteit Leuven, Belgium

Abstract.

Most tasks in DSP-applications require multiple resources for their execution. If only CPU-usage is considered while constructing a static schedule, the actual run-time performance of the application can differ a lot from the predicted one. In this paper we present a scheduling method that takes non-CPU resource-requirements into account as well while constructing the static schedule.

Situation of the problem.

Most Digital Signal Processing (DSP) applications have to operate at very high sample-rates (order of MHz), such that *static schedules* are required to obtain real-time performance. Such static schedules are mostly automatically produced by CAD-tools [3], [5], [6], [7]. Most of these tools consider CPU-usage to determine a "time-optimal" task-execution order at compile-time. Once the order is fixed, code is generated and downloaded on the processor hardware for execution.

During the execution of such a schedule however, it is quite possible that the actual performance differs a lot from what was predicted by the static schedule. First it is possible that the run-time makespan far exceeds the estimated one. This might be caused by wrong timing-estimates for the tasks in the application. When the estimate supposes e.g. the use of internal memory to store program-code and data, while the actual task uses external memory, the real execution-time greatly exceeds the estimated one. But even if all estimates of execution-times are accurate, the actual makespan can still differ a lot from the predicted one. This happens when multiple tasks try to access a shared device like e.g. a communication-link or a shared bus at the same time. In this case only one task gets access to the device while the others have to wait until it is released again. Such contention-delays are not counted for in a classically constructed static schedule.

Besides those timing-aspects, also other surprises can arise while trying to execute the schedule. When too many tasks are assigned to the same device, the total amount of memory required can exceed the available amount. In this case we do not even succeed in downloading the code on the multi-processor board and the actual execution can never start. Another problem arises when the same hardware FIFO-buffer is used for communication between multiple tasks. When the first write-operation corresponds e.g. with the communication between two tasks A and B, while the first scheduled read-operation delivers data to a task C, the application will behave incorrectly.

Resource-constraint scheduling in GRAPE-II.

To avoid these problems and to guarantee that the static schedule we produced behaves as we expected, we have to take resource-requirements into account. In GRAPE-II (Graphical RApid Prototyping Environment) [1], [2], [4] this is done by representing all resource-requirements as separate tasks that need to be assigned to the different resources. Every user-task then corresponds to a cluster of subtasks, one for each associated resource-requirement. During assignment an entire task-cluster is assigned to a cluster of devices. The router adds communication-tasks, wherever data need to be transported. And finally in the scheduler all subtasks are ordered on their devices. To preserve the timing-relations between related subtasks, all subtasks belonging to the same task-cluster are placed together, taking into account their internal timing-relations as well as the load on the devices they have to be executed on.

Depending on the nature of the resource-requirement, three types of resource-subtasks are distinguished.

First there are resources that are required for a fixed time that depends on the execution of a CPU-subtask. Examples of such resources are A/D- and D/A-converters, communication-ports etc. Those resource-requirements are represented as fixed size subtasks and are assigned and scheduled in the same way as a normal CPU-task.

Other resources are required for a time that depends on the final schedule. This occurs when one (CPU-) task starts the resource-use while the execution of another one releases the resource again. A typical example can be found in the use of FIFO-buffers for communication purposes. Such a buffer is required from the moment the sender-task puts data in it, until the moment the receiver has read the data out of it. When sending and receiving actually occur depends on the schedule and is not fixed beforehand. To represent such resource-requirement of a priori unknown duration, we make use of an elastic subtask. The start of such an elastic has a strict timing-relationship with one cluster of fixed size subtasks, while the end of it is coupled with another (independent) task-cluster. For assignment purposes both start- and end-related clusters are supposed to form one hierarchical cluster together with the elastic subtask. This makes it easier to satisfy the required inter-device connections. During scheduling however the hierarchical cluster is split again in its set of subclusters. Together with the corresponding start-cluster the start of an elastic subtask is scheduled and keeps the resource busy until also the corresponding end-cluster has been scheduled. From that moment on the resource is released again for use by other tasks.

Finally there also exist resources that are required permanently as long as the application runs, like program-memory. Since no run-time congestion can occur on such a device, they do not need to be considered during scheduling. During assignment however, they are of major importance to check if a cluster of temporary subtasks, can actually be assigned to the preferred device-cluster or not. The amount of free program-memory left, could for instance be less than what is required by the task. If this is the case another device-cluster for temporarily use needs to be searched for.

Using this subtask representation most of the problems mentioned before are solved. By basing the execution-time estimates on the assignment of all cluster-subtasks, these are much more accurate and will clearly lead to more realistic performance-estimates. The scheduling of subtasks on all resources avoids the problem of unexpected resource-contention. The problem of device-overloading on its turn is avoided by only assigning tasks to devices when there is enough free place left to add the task under consideration as well.

The problem of incorrect FIFO-buffer management however still needs some special attention. To guarantee that data are read in the same order as they were written, we make use of dynamic sequence-edges. Such sequence-edge connects the reading parts of two tasks that use the same FIFO-buffer, in the order their writing parts were scheduled. This guarantees that data will automatically be consumed by the task they are meant for.

Acknowledgements.

Greet Bilsen is a Research Assistant and Rudy Lauwereins a Senior Research Associate of the Belgian National Fund for Scientific Research. K.U.Leuven-ESAT is a member of the DSP-Valley network. This project is partially sponsored by the Belgian Interuniversity Pole of Attraction IUAP-50 and the ESPRIT project 6800 Retides.

References.

[1] G. Bilsen, P. Wauters, M. Engels, R. Lauwereins and J.A. Peperstraete, "Development of a static load balancing tool", Proceedings of the Fourth Workshop on Parallel and Distributed Processing '93, pp. 179-194, Sofia, Bulgaria, May 4-7, 1993.

[2] G. Bilsen, M. Engels, R. Lauwereins and J.A. Peperstraete, "Development of a tool for compile-time assignment on a multi-processor," Internal Report, Katholieke Universiteit Leuven, *ESAT-Laboratory,* May 1993.

[3] P. Hoang, J. Rabaey, "Partitioning of DSP Programs onto Multiprocessors for Maximum Throughput", Internal Report, Electronics Research Laboratory, University of California, Berkeley, 26 April 1991.

[4] Rudy Lauwereins, Marc Engels, Marleen Adé, J.A. Peperstraete, "GRAPE-II: Graphical RApid Prototyping Environment for Digital Signal Processing Systems", *proceedings of ICSPAT '94*, Oct. 18-21, 1994, Dallas, Texas.

[5] S. Note, P. Vandebroeck, P. Odent, D. Genin, M. Van Canneyt, "Top Down design of two industrial IC's with DSP Station, DSP Application, 1993.

[6] J. Pino, S. Ha, E. Lee, J. Buck, "Software Synthesis for DSP Using Ptolemy", Journal on VLSI Signal Processing, special issue on Synthesis for DSP, 1993.

[7] Signal Processing Worksystem (SPW ™) Manuals, Comdisco Systems, Inc.

Proceedings of the 28th Annual Hawaii International Conference on System Sciences — 1995

On Optimal Strategies for Stealing Cycles

Sandeep Bhatt[1] Fan Chung[1] Tom Leighton[2] Arnold Rosenberg[3]

[1] Bellcore, Morristown NJ 07960.

[2] MIT, Cambridge MA 02139.

[3] U. Massachusetts, Amherst MA 01003.

Abstract. The growing importance of networked workstations as a computing milieu has created a new modality of parallel computing, namely, the possibility of having one workstation *"steal cycles"* from another. In a typical episode of cycle-stealing, the owner of workstation B allows the owner of workstation A to take control of B's processor whenever it is idle, with the promise of relinquishing control immediately upon the demand of the owner of B. Typically, the costs for an episode reside in the overhead in transmitting work, coupled with the fact that work in progress when the owner of B reclaims the workstation is lost to the owner of A. The first cost militates toward supplying B with a large amount of work at once; the second cost militates toward repeatedly supplying B with small amounts of work. This paper formulates a model of cycle-stealing and studies strategies that optimize the expected work from a single episode.

1 Motivation

Research on parallel computing has historically focussed on single machines that are endowed with many processors. The growing importance of networked workstations as a computing milieu has created an alternative modality of parallel computing, namely, the process of *stealing cycles*; see [1-6]. The following scenario defines this paradigm. The owner of workstation A has a massive number of mutually independent tasks that must be computed. In order to expedite the completion of the tasks, the owner of A enters a contract with the owners of (some of) the other workstations in the cluster, that allows A to take control of the processors of these workstations whenever they are idle, with the promise of relinquishing control *immediately* upon the demand of a workstation's owner (say, when the mouse or keyboard is touched). The question studied here is: When workstation B becomes available, how should the owner of A allocate work to B in order to maximize the total amount of work one can expect to garner from a single episode of cycle-stealing? The challenge of this problem resides in the tension created by the main costs of an episode of cycle-stealing. The first cost resides in the *fixed* portion of the overheads of supplying work to workstation B and reclaiming the results of that work: in a data-parallel situation, these fixed overheads would reside in "filling the pipe" twice, first to supply input data to B and second to receive output data from B; in the most general situation, these overheads would also include the cost of supplying B with the appropriate programs. We ignore for the moment the second, *variable*, component of the cost of supplying B with work, i.e., the per-datum portion of the cost, for in our model, we absorb this variable cost into the cost of B's executing the assigned tasks. The third component of the cost resides in the risk that the owner of A will lose the results of whatever work is in progress when B's owner reclaims that workstation — due to the promise to abandon B *immediately* upon demand. The first cost would lead the owner of A to give B a single large package of tasks; the third cost would lead the owner of A to give B a sequence of small packages of tasks. Clearly, the owner of A must seek a strategy that balances the first and third costs in a way that maximizes the expected return from an episode. We formulate a mathematical model of the process of cycle-stealing and study strategies that optimize, under a variety of assumptions, this expected return.

2 A Mathematical Model of Cycle-Stealing

2.1 The Relevant Notions

Lifespans. Clearly, no single strategy can suffice for all possible episodes of cycle-stealing: as an extreme example, an episode arising from a multi-week vacation must be treated differently from an episode

arising from a telephone call. Accordingly, we consider two *scenarios*, or classes of episodes, that require somewhat different groundrules. Within these scenarios, we allow different probability distributions on the "risk" of the return of the owner of workstation B. In the *unbounded lifespan* scenario, the owner of A has no *a priori* upper bound on how long workstation B will remain idle; in the *bounded lifespan* scenario, the owner of A knows that workstation B will be idle for at most L time units.[1] In both scenarios, the owner of A is given information about the *a priori* probability distribution on the "risk" of the return of the owner of B. In Sections 3-5, we assume total information about the distribution; in Section 6, we assume no information is available.

Work Schedules. The owner of A partitions the lifespan of B into a *schedule*, i.e., a sequence $\mathcal{S} = t_0, t_1, t_2, \ldots$ of finite-length *periods* (the ith period having length $t_i \geq 0$), with the following intension. At time τ_k, the kth period begins, and the owner of A supplies B with an amount of work chosen[2] so that t_k time units are sufficient for

- the owner of A to send the work to B,
- B to perform the work,
- B to return the results of the work.

If $k = 0$, then $\tau_k = 0$; if $k > 0$, then $\tau_k = T_{k-1} =_{\text{def}} t_0 + t_1 + \cdots + t_{k-1}$.

Communication Costs. The communication that starts and ends each period in an episode incurs a fixed *overhead* of c time units. This overhead results from some combination of:

- A sending B a message "telling it" where to get data and/or programs;
- B accessing a storage device to get data and/or programs, or to return results;
- A "filling the pipe" while sending B data and/or programs (from its local memory);
- B "filling the pipe" while returning results.

Of course, programs could be prestored in all workstations, especially in data-parallel computations.[3] The fact that c is typically large compared to the per-task computing time would lead the owner of A to try to minimize the number of periods in a schedule.

[1] For instance, L might be a night, a 24-hour day, or a week.

[2] We assume here that task lengths are known exactly. In later work, we shall weaken this assumed knowledge.

[3] The fact that c is independent of how much work is allocated to B means that our model includes in computing time the marginal pipelined costs of transmitting data to and receiving data from B.

Work Schedules Revisited. At time τ_k, the beginning of period k of schedule $\mathcal{S} = t_0, t_1, t_2, \ldots$, A supplies B with[4] $w_k =_{\text{def}} t_k \ominus c$ units of work. If the owner of B has not returned by time $T_k = \tau_k + t_k$, then the amount of work done so far during this episode is augmented by w_k; if the owner has returned by time T_k, then the episode terminates, with the total amount of work $w_0 + w_1 + \cdots + w_{k-1}$ (so work that is interrupted by the return of the owner of B is lost). Two facts emerging from this scenario should be kept in mind:

1. Because of the overhead c, a period of length t produces at most $t \ominus c$ work.

2. In the *bounded lifespan* scenario with lifespan L, the risk of being interrupted, hence losing work, may make it desirable to have the productive t_i (those exceeding c) sum to *less than* L.

Risks. The *risk* in an episode is characterized by the nonincreasing *life function* p: $p(t)$ is the probability that the owner of B *has not* returned by time t. p is defined formally via the *risk function* q which gives the probability that the owner of B returns at precisely time t:

$$p(t) = 1 - \int_{i=0}^{t} q(i)di.$$

We perform our study in a continuous rather than discrete domain to simplify certain manipulations. The reader should easily be able to extract discrete approximations to our results.

Expected Work. Our goal throughout is to maximize the *expected work* in an episode. Under schedule $\mathcal{S} = t_0, t_1, \ldots$ and life function p, this quantity is denoted $E(\mathcal{S}; p)$ and is given by

$$E(\mathcal{S}; p) = \sum_{i \geq 0} (t_i \ominus c) p(T_i) = \sum_{i \geq 0} w_i p(T_i). \quad (1)$$

The summation in (1) must account for every period in schedule $\mathcal{S}$. Accordingly, its upper limit is ∞ in the *unbounded lifespan* scenario and $m - 1$ in an m-period *bounded lifespan* scenario.

The framework we have developed makes the cycle stealing problem formally similar to the problem of allocating work to a system that can fail catastrophically. However, the models differ in details and the techniques used in the analysis differ significantly from those that are usually brought to bear on the work allocation problem [3].

[4] The operator "$\ominus$" denotes *positive subtraction:* $x \ominus y =_{\text{def}} \max(0, x - y)$.

Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)
1060-3425/95 $10.00 © 1995 IEEE

2.2 A Simplifying Observation

A schedule $\mathcal{S} = t_0, t_1, \ldots$ is *optimal* for life function p if $E(\mathcal{S}; p) \geq E(\mathcal{S}'; p)$ for any other schedule $\mathcal{S}'$. The following technical lemma simplifies our quest for optimal schedules by showing that *nonproductive periods*, i.e., those whose lengths do not exceed the communication overhead c, hence contribute no work, cannot occur frequently. Specifically, a nonproductive period cannot occur at all in an optimal schedule for the *unbounded lifespan* scenario, and one can occur only as period $m-1$ in an m-period *bounded lifespan* scenario.

Lemma 1 *For every episode of cycle-stealing with communication overhead c, there is an optimal schedule $\mathcal{S}$ that is* productive, *in the following sense. If $\mathcal{S}$ has infinitely many periods (in the unbounded lifespan scenario), then every period of $\mathcal{S}$ has length $> c$. If $\mathcal{S}$ has m periods (in the bounded lifespan scenario), then every period of $\mathcal{S}$, save possibly the last, has length $> c$.*

Proof. Note first that we can lose no generality by assuming that all periods in a schedule have *positive* length. Let us focus, therefore, on a schedule $\mathcal{S} = t_0, t_1, \ldots, t_k, t_{k+1}, \ldots$, where $0 < t_k \leq c$. Construct the schedule $\mathcal{S}^{<k>} = s_0, s_1, \ldots$ from $\mathcal{S}$ as follows. If $\mathcal{S}$ has infinitely many periods, then so also does $\mathcal{S}^{<k>}$; if $\mathcal{S}$ has m periods, then $\mathcal{S}^{<k>}$ has $m-1$ periods; in either case, the periods of $\mathcal{S}^{<k>}$ are defined by:

$$s_i = \begin{cases} t_i & \text{for } i < k \\ t_k + t_{k+1} & \text{for } i = k \\ t_{i+1} & \text{for } i > k \end{cases}$$

We claim that $E(\mathcal{S}^{<k>}; p) \geq E(\mathcal{S}; p)$ for all life functions p. To wit, by direct calculation:

$$\begin{aligned} & E(\mathcal{S}^{<k>}; p) - E(\mathcal{S}; p) \\ = \; & (t_k + t_{k+1} \ominus c)p(T_k + t_{k+1}) - (t_k \ominus c)p(T_k) \\ & - (t_{k+1} \ominus c)p(T_k + t_{k+1}) \\ = \; & [(t_k + t_{k+1} \ominus c) - (t_{k+1} \ominus c)]p(T_k + t_{k+1}) \\ \geq \; & 0. \end{aligned}$$

In other words, if a positive-length nonproductive period appears as any but the last period of $\mathcal{S}$, one can never decrease the expected work of $\mathcal{S}$ by combining the nonproductive period with its successor. □

3 The Geometrically Decreasing Lifespan Model

We begin our study with the *geometrically decreasing lifespan* (*GDL*) model, wherein each episode of cycle-stealing has a "half-life;" i.e., the probability that an episode lasts at least $\ell + 1$ time units is roughly half the probability that it lasts at least ℓ time units. This model fits most naturally within the *unbounded lifespan* scenario. For the sake of generality and reality, we replace the parameter $1/2$ in "half-life" by $1/a$ for some *risk parameter* $a > 1$. This adds a bit of realism, in the sense that the "half-life" of an episode need not be measured in the same time units as is work. Note that, with any given risk factor, the conditional distribution of risk in this model looks the same at every moment of time. This fact will enter implicitly into our analysis of the model.

Formally, the life function for the GDL model with risk parameter a is given by:

$$p_a(t) = a p_a(t+1) = a^{-t}$$

for all $t \geq 0$.

It is not clear *a priori* that there exist schedules that are optimal for the GDL model.[5] We now prove that optimal schedules do exist and that they can be chosen to be *uniform*, in the sense of having equal-length periods.

Theorem 1 *The following uniform schedule, $\mathcal{S}^{(a)}$, is optimal for the GDL model with risk parameter a. $\mathcal{S}^{(a)}$ has periods of common length $t^{(a)}$ defined implicitly by the equation*[6]

$$t^{(a)} \ln a + a^{-t^{(a)}} = 1 + c \ln a. \tag{2}$$

$\mathcal{S}^{(a)}$ has expected work

$$E(\mathcal{S}^{(a)}; p_a) = \frac{a^{-t^{(a)}}}{\ln a}. \tag{3}$$

Proof Sketch. Our first task is to verify that there exists an optimal schedule for the GDL model with risk parameter a. This follows from the Least Upper Bound Principle via the following reasoning. Let $\mathcal{S} = t_0, t_1, \ldots$ be a schedule all of whose periods have length $> c$. (By Lemma 1, if there exists an optimal schedule, then there exists such a productive one.) By definition, then,

$$\begin{aligned} E(\mathcal{S}; p_a) &= \sum_{i=0}^{\infty} (t_i - c) a^{-T_i} \\ &\leq \sum_{i=0}^{\infty} (t_i - c) a^{-(t_i + T_{i-1})} \\ &\leq \sum_{i=0}^{\infty} (t_i - c) a^{-(t_i + (i-1)c)}. \end{aligned}$$

[5] This is a real concern: For instance, the life function $p(t) = 1/(t+1)$ does not admit an optimal schedule.

[6] $\ln a$ denotes the natural logarithm of a.

Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)
1060-3425/95 $10.00 © 1995 IEEE

Since the form xa^{-x} is bounded above by a constant, it follows that $E(\mathcal{S};p_a)$ is also bounded above by a constant, whence the claim.

Next, consider the *tail* of $\mathcal{S}$, i.e., the schedule $\mathcal{S}' = t_1, t_2 \ldots$. If $E(\mathcal{S}';p_a) > E(\mathcal{S};p_a)$, then we abandon $\mathcal{S}$ and concentrate instead on $\mathcal{S}'$. Otherwise, we have $E(\mathcal{S};p_a) \geq E(\mathcal{S}';p_a)$, so that

$$\begin{aligned} E(\mathcal{S};p_a) &= (t_0 - c)a^{-t_0} + a^{-t_0}E(\mathcal{S}_1;p_a) \\ &\leq (t_0 - c)a^{-t_0} + a^{-t_0}E(\mathcal{S};p_a). \end{aligned}$$

By solving this recurrence, we obtain the bound.

$$E(\mathcal{S};p_a) \leq \frac{t_0 - c}{a^{t_0} - 1}. \tag{4}$$

Now, it turns out that the uniform schedule $\mathcal{S}'' = t, t, \ldots$ has expected work

$$E(\mathcal{S}'';p_a) = \frac{t - c}{a^t - 1}, \tag{5}$$

which matches the bound of inequality (4). It follows that there is a uniform schedule, call it $\mathcal{S}^{(a)}$, that is optimal for the GDL model with risk parameter a; and, we can find $\mathcal{S}^{(a)}$ by determining the value $t^{(a)}$ of t that maximizes expression (5). This determination is a straightforward calculus exercise that leads to equations (2) and (3). The Theorem follows. □

In the full paper [1]: (*a*) we prove that $\mathcal{S}^{(a)}$ is the *unique* optimal schedule; (*b*) we prove a weak converse to Theorem 1, showing that any cycle-stealing episode for which a uniform schedule is optimal honors a weak analog of the GDL model.

4 The Uniform Risk Model

We turn next to a *bounded lifespan* scenario. In the *uniform risk* (*UR*) model, the probability of "dying" is the same at every moment. The intention is to model a situation wherein one can predict with some confidence that the owner of workstation B is likely to be absent for at most L time units, but wherein there is a slowly growing probability that (s)he will return early, so that the probability of being alive at any particular time decreases by a fixed constant with each passing time unit. A similar model, with similar results, is studied in [3] but using different techniques.

The life function for the UR model with lifespan L is given by $p_L(t) = 1 - t/L$ for $0 \leq t \leq L$ (so the risk function is given by $q_L(t) \equiv 1/L$).

The unique optimal schedule for the UR model partitions the lifespan into periods that decrease in length arithmetically , with common difference c (the communications overhead), and that are maximal in number, given this rate of decrease.

Theorem 1 *The unique optimal schedule for the UR model with lifespan L consists of m periods: $\mathcal{S}^{(\text{opt})} = t_0^{(\text{opt})}, t_1^{(\text{opt})}, \ldots, t_{m-1}^{(\text{opt})}$, where*

$$m = \left\lfloor \sqrt{\frac{2L}{c} + \frac{1}{4}} - \frac{1}{2} \right\rfloor; \tag{6}$$

for $0 \leq i < m$,

$$t_i^{(\text{opt})} = \frac{L}{m+1} + \frac{cm}{2} - ci. \tag{7}$$

The resulting expected work is

$$\begin{aligned} E(\mathcal{S}^{(\text{opt})};p_L) &= \frac{L}{2} - mc - \frac{L}{2(m+1)} \\ &\quad + \frac{m(m+1)(m+2)c^2}{24L}. \end{aligned} \tag{8}$$

Proof Sketch. Let $\mathcal{S} = t_0, t_1, \ldots, t_{m-1}$ be an m-period candidate for an optimal schedule for the UR model with lifespan L. Note that the number of periods m is unknown here, as well as the period lengths. When equation (1) is instantiated with schedule $\mathcal{S}$ and risk function p_L, in the light of Lemma 1, it can be written in the form

$$\begin{aligned} &\sum_{i=0}^{m-2}(t_i - c)\left(1 - \frac{1}{L}T_i\right) + (t_{m-1} \ominus c)\left(1 - \frac{1}{L}T_{m-1}\right) \\ = &\sum_{i=0}^{m-2}(t_i - c)\left(1 - \frac{1}{L}T_i\right). \end{aligned} \tag{9}$$

We uncover the parameters of the optimal schedule for the UR model with lifespan L by deriving an assignment of values to the variables $m, t_0, t_1, \ldots, t_{m-1}$, that maximizes expression (9) subject to the positivity of c and L, the positive integrality of m, and the constraints:

- $t_i > c$ for $0 \leq i < m-1$ (because of Lemma 1);
- $\sum_{i=0}^{m-1} t_i = L$.

In fact, we lose no generality if we replace expression (9) by the expression

$$\sum_{i=0}^{m-1}(t_i - c)\left(1 - \frac{1}{L}T_i\right), \tag{10}$$

with the constraint on the t_i weakened to

- $t_i \geq 0$ for $0 \leq i \leq m-1$.

Although this change broadens the search space for

Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)
1060-3425/95 $10.00 © 1995 IEEE

a maximizing assignment, the maximizing assignment for expression (10) actually satisfies our demands.

We begin our search for a maximizing assignment by rewriting expression (10) to better expose its fundamental structure. By direct manipulation, plus the fact that $L^2 = (t_0 + t_1 + \cdots + t_{m-1})^2$, we find that

$$\begin{aligned} & L \cdot \sum_{i=0}^{m-1} (t_i - c)(1 - T_i/L) \\ = & \sum_{i=0}^{m-1} (t_i - c)(t_{i+1} + t_{i+2} + \cdots + t_{m-1}) \\ = & \frac{1}{2}L^2 - \frac{1}{2}\sum_{i=0}^{m-1} \left[(t_i + ci)^2 - c^2 i^2\right]. \end{aligned}$$

Letting $u_i = t_i + ci$ for $0 \le i < m$, we convert the last expression to the perspicuous expression for $E(\mathcal{S})$ that we shall actually maximize.

$$E(\mathcal{S}) = \frac{L}{2} + \frac{c^2 m(m-1)(2m-1)}{12L} - \frac{1}{2L}\sum_{i=0}^{m-1} u_i^2. \quad (11)$$

Our task now is to maximize expression (11) subject to the following constraints.

1. Since each $t_i \ge 0$, we must have each $u_i \ge ci$.
2. Since $\sum_{i=0}^{m-1} t_i = L$, we must have

$$\sum_{i=0}^{m-1} u_i = \sum_{i=0}^{m-1} t_i + ci = L + c\frac{m(m-1)}{2}.$$

Easily, we shall have achieved our goal once we have minimized the sum $u_0^2 + u_1^2 + \cdots + u_{m-1}^2$ subject to the same constraints. Now, were it not for constraint (1), we could minimize this sum simply by setting each u_i to its average value

$$u_i = \frac{L}{m} + \frac{c(m-1)}{2}.$$

Constraint (1) forces us to be a bit more careful. Specifically, we can use this simple averaging only for the first $r+1$ terms of the sum, where r is the largest integer such that

$$\frac{L}{r+1} + \frac{cr}{2} \ge cr. \quad (12)$$

For the remainder of the sum, we use the value forced on us by constraint (1), namely $u_i = ci$. We thus end up with the following minimizing assignment for the u_i:

$$u_i = \begin{cases} L/(r+1) + cr/2 & \text{for } i \le r \\ ci & \text{for } i > r \end{cases} \quad (13)$$

The desired maximizing value of r is obtained by rewriting expression (12) in the form $r^2 + r - 2L/c \le 0$. It is now clear that the maximizing value of r is given by

$$r = \left\lfloor \sqrt{\frac{2L}{c} + \frac{1}{4}} - \frac{1}{2} \right\rfloor. \quad (14)$$

The minimizing assignment (13) to the variables $\{u_i\}$ yields the sought maximizing assignment to the variables $\{t_i\}$:

$$t_i = \begin{cases} L/(r+1) + cr/2 - ci & \text{for } i \le r \\ 0 & \text{for } i > r \end{cases} \quad (15)$$

Since all periods beyond the first r have zero length, hence contribute no work, the maximizing value of r in equation (11) is in fact the sought number of periods m in the optimal schedule, thus verifying equation (6). Assignment (15) directly yields the sought values for the period lengths $\{t_i\}$, thus verifying equation (7).

We complete the proof by verifying that our maximizing assignments for m and $\{t_i\}$ yield equation (8). Direct evaluation of expression (11), with the maximizing values of all parameters accomplishes this. Details are left to the reader. □

For the sake of perspective, we note that $E(\mathcal{S}^{(\text{opt})}; p_L)$ is very close to

$$E(\mathcal{S}^{(\text{opt})}; p_L) = \frac{L}{2} - \frac{2}{3}\sqrt{2cL}$$

when L is very much larger than c (which is likely to be the case).

5 The Geometrically Increasing Risk Model

As our final example, we consider the *geometrically increasing risk* (*GIR*) model, a *bounded lifespan* model which may be appropriate when the owner of workstation B is likely to be absent for only a short period of time, say because of a telephone call. In this model, the probability of B's owner's return doubles at each time unit. We choose to interpret "double" literally here (in contrast to Section 3) to keep the arithmetic simplicity of a singly parameterized model.

The life function for the GIR model with lifespan L is given by $p_L^g(t) = (2^L - 2^t)/(2^L - 1)$ for $t \ge 0$ (which corresponds to a discrete risk function $q_L^g(t+1) = 2q_L^g(t) = (2^L - 1)2^{-(t-1)}$ for $t \ge 1$).

The optimal schedule for the GIR model partitions the lifespan into periods of exponentially decreasing lengths.

Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)
1060-3425/95 $10.00 © 1995 IEEE

Theorem 1 *The optimal schedule for the GIR model has m periods: $\mathcal{S}^{(\mathrm{opt})} = t_0^{(\mathrm{opt})}, t_1^{(\mathrm{opt})}, \ldots, t_{m-1}^{(\mathrm{opt})}$, where, to within rounding,*[7]

$$m = \log^\star L - \log^\star c, \tag{16}$$

and where the $t_i^{(\mathrm{opt})}$ are given implicitly by

$$t_k^{(\mathrm{opt})} = 2^{t_{k+1}^{(\mathrm{opt})}} + c - 2 \quad \text{for} \quad 0 \le k \le m-2. \tag{17}$$

Proof Sketch. Let $\mathcal{S} = t_0, t_1, \ldots, t_{m-1}$ be a candidate optimal m-period schedule for the GIR model with lifespan L. Instantiating equation (1) for the life function p_L^g and invoking Lemma 1, we find, after some manipulation, that

$$E(\mathcal{S}; p_L^g) = \frac{1}{2^L - 1}\left(2^L(L-1) - mc2^L - \sum_{i=0}^{m-1} w_i 2^{T_i - 1}\right).$$

We use a perturbation argument to analyze schedule $\mathcal{S}$. The *kth-period positive (resp., negative) perturbation* of $\mathcal{S}$, denoted $\mathcal{S}^{+k}$ (resp., $\mathcal{S}^{-k}$) is the schedule

$$\mathcal{S}^{\pm k} =_{\mathrm{def}} t_0, t_1, \ldots, t_{k-1}, t_k \pm 1, t_{k+1} \mp 1, t_{k+2}, \ldots, t_{m-1}.$$

Note that these both have the same lifespan L as does $\mathcal{S}$.

If schedule $\mathcal{S}$ were optimal, then its expected work would be no less than that of any of its perturbations. By applying this observation to both $E(\mathcal{S}) - E(\mathcal{S}^{+k})$ and $E(\mathcal{S}) - E(\mathcal{S}^{-k})$, we verify equation (17).

To verify equation (16), we recall (from Lemma 1) that each period of an optimal finite schedule, save, perhaps the last, must have length $> c$; hence, we can continue to take the logarithm of $(w_0 = t_0 - c)$, then of

$$w_1 = t_1 - c = \log(w_0 + 2),$$

then of

$$\begin{aligned} w_2 &= t_2 - c \\ &= \log(w_1 + 2) - c \\ &= \log(t_1 - c + 2) - c \\ &= \log(\log(w_0 + 2) - c + 2) - c, \end{aligned}$$

and so on, only so long as we attain periods of length $t > 2^c + c - 2$, at which point additional periods would not be productive. □

[7] Throughout this section, all logarithms are to the base 2. If we inductively let $\log^{(i+1)} x = \log(\log^{(i)} x)$, then $\log^\star x$ denotes the smallest integer r for which $\log^{(r)} x \le 1$.

6 Operating with an Unknown Life Function

We consider finally a scenario wherein very little is known about the life function. Of course, if nothing at all is known about the life function, then we can do little with confidence, since our first task can be killed just before it is finished (no matter how long the task is). The model we study now assumes only that we know that in a window of L units of time (possibly containing many interrupts), an optimal prescient scheduling strategy can accomplish αL units of work. Our goal is to design a strategy that accomplishes βL units of work where β is as close to α as possible. In a typical application, it is reasonable to assume that α is close to one; let us conservatively assume that $\alpha > 1/3$.

This scenario differs from those considered earlier in several respects. First, we now allow epsiodes to include many interrupts, not only one. Second, we assume nothing about the life function during an interval between interrupts, except that over the entire L time units, one could accomplish αL work if one could anticipate the interrupts. Third, we judge performance by comparison with an optimal prescient schedule.

We first study the problem of devising a deterministic oblivious strategy. "Oblivious" here means that the length of the ith task is independent of where prior interrupts (if any) occurred. The ith task is started as soon as the $(i-1)$th task has finished and/or once the workstation becomes available following an interrupt during the $(i-1)$th task. (Similar results hold for the model in which the $(i-1)$th task is restarted if it is interrupted before the ith task is run.)

Let L_i be length of the ith interval of available workstation time for $0 \le i < m$, where m is the number of available intervals in the entire window of L steps. Without loss of generality, $L_i > c$ for each i. Since the optimal prescient schedule achieves αL work, we know that

$$\sum_{i=0}^{m-1} (L_i - c) \ge \alpha L.$$

We know also that $\alpha L + cm \le L$ and thus that $m \le (1-\alpha)L/c$.

Now, consider a schedule for which every task has the same length $t > c$. (We show that such a strategy is, in some sense, optimal.) The work achieved by this schedule is easily

$$\sum_{i=0}^{m-1} \lfloor L_i/t \rfloor (t-c) \quad \ge \quad (t-c) \sum_{i=0}^{m-1} (L_i/t - 1)$$

$$\geq (t-c)[\alpha L - (t-c)m]/t$$
$$\geq (t-c)(c-t+\alpha t)L/tc.$$

The preceding expression is optimized by setting $t = c/\sqrt{1-\alpha}$, whereupon the work achieved is $(1-\sqrt{1-\alpha})^2 L$, which provides a competitive ratio of $(1-\sqrt{1-\alpha})/(1+\sqrt{1-\alpha})$ over the optimal prescient schedule. For α close to 1, the work achieved is close to L, and the competitive ratio is close to 1. For instance, when $\alpha > 3/4$, the competitive ratio is greater than $1/3$.

Using the observation that an adversary's best strategy is to kill the $\lfloor(1-\alpha)/c\rfloor$ longest running tasks, one can show that the preceding schedule is optimal among deterministic oblivious schemes when it comes to maximizing the amount of work that we are guaranteed to be able to achieve.

Devising an optimal *adaptive* strategy — one where the task lengths are chosen based on history — is a more difficult matter. For example, we now consider a scenario in which we are assured that there will be at most one interrupt (of known length). Without loss of generality, we assume that the interrupt has length zero. This situation corresponds to the scenario where $\alpha L > L - 3c$. The optimal schedule for this scenario is as follows.

Define i to be the (not necessarily unique) integer such that

$$\frac{(i-1)i}{2} \leq \frac{L}{c} - 1 \leq \frac{i(i+1)}{2}.$$

Define

$$t_j = \frac{L + (i^2+i-2)c/2}{i} - jc$$

for $1 \leq j < i$, and set $t_i = t_{i-1}$. Then the optimal schedule is to run a task of length t_1, followed by a task of length t_2, followed by a task of length t_3, and so on, until there is an interrupt, after which we run a single task that consumes the remaining time. (Easily, $t_1 + t_2 + \cdots + t_i = L$.) One can show that the work accomplished by this schedule is at least

$$\frac{(i-1)L - (i^2+i-2)c/2}{i},$$

no matter where the interrupt occurs. One can also prove by induction that this strategy is optimal if only one interrupt can occur. (Details will appear in the final version of the paper.)

It is worth noting that the strategy just described is very similar to the optimal strategy for the UR model (Section 4). Indeed, when $L >> c$, we begin with a task of length about $\sqrt{2L/c}$, and then select tasks with lengths that successively decrease by c in length until the interrupt occurs. Hence, the optimal strategy against an unknown interrupt is very similar to the strategy where we assume that the interrupt will be uniformly distributed, although the two scenarios are somewhat different.

We close with three open problems concerning the framework of this section. The first problem is to find a closed-form solution for an optimal strategy when two or more interrupts are allowed. The second is to devise optimal randomized strategies. The third is to verify or refute our conjecture that the competitive performance of randomized strategies is better than that of deterministic strategies.

Acknowledgments

The authors would like to thank David Kaminsky and David Gelernter for discussions that got us started on this work. The research of F. T. Leighton was supported in part by Air Force Contract OSR-86-0076, DARPA Contract N00014-80-C-0622; the research of A. L. Rosenberg was supported in part by NSF Grants CCR-90-13184 and CCR-92-21785. A portion of this research was done while F. T. Leighton and A. L. Rosenberg were visiting Bell Communications Research.

References

[1] S. Bhatt, F. Chung, T. Leighton and A. Rosenberg, Optimal strategies for stealing cycles, in preparation, 1994.

[2] D. Cheriton (1988): The V distributed system. *C. ACM*, 314-333.

[3] E.G. Coffman, L. Flatto and A. Y. Kreinin, Scheduling saves in fault-tolerant computations, Acta Informatica, 30(1993), 409-423.

[4] D. Gelernter and D. Kaminsky (1991): Supercomputing out of recycled garbage: preliminary experience with Piranha. Tech. Rpt. RR883, Yale Univ.

[5] M. Litzkow, M. Livny, M. Matka (1988): Condor - A hunter of idle workstations. *8th Ann. Intl. Conf. on Distributed Computing Systems.*

[6] D. Nichols (1990): *Multiprocessing in a Network of Workstations.* Ph.D. thesis, CMU.

[7] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelsom, B. Welch (1988): The Sprite Network Operating System. *IEEE Computer 21*, 6, 23-36.

[8] A. Tannenbaum (1990): Amoeba: a distributed operating system for the 1990s. *IEEE Computer*, 44-53.