Static Strategies for Worksharing with Unrecoverable Interruptions

A. Benoit[2,4,5] Y. Robert[2,4,5] A. L. Rosenberg[1] F. Vivien[3,4,5]

[1] Colorado State University, USA [2] ENS Lyon, France [3] INRIA, France

[4] Université de Lyon, France [5] LIP, UMR 5668 ENS-CNRS-INRIA-UCBL, Lyon, France

Abstract

One has a large workload that is "divisible"—its constituent work's granularity can be adjusted arbitrarily—and one has access to p remote computers that can assist in computing the workload. The problem is that the remote computers are subject to interruptions of known likelihood that kill all work in progress. One wishes to orchestrate sharing the workload with the remote computers in a way that maximizes the expected amount of work completed. Strategies for achieving this goal, by balancing the desire to checkpoint often, in order to decrease the amount of vulnerable work at any point, vs. the desire to avoid the context-switching required to checkpoint, are studied. Strategies are devised that provably maximize the expected amount of work when there is only one remote computer (the case $p = 1$). Results suggest the intractability of such maximization for higher values of p, which motivates the development of heuristic approaches. Heuristics are developed that replicate works on several remote computers, in the hope of thereby decreasing the impact of work-killing interruptions. The quality of these heuristics is assessed through exhaustive simulations.

1 Introduction

Technological advances and economic constraints have engendered a variety of modern computing platforms that allow a person who has a massive, compute-intensive workload to enlist the help of others' computers in executing this workload. The cooperating computers may belong to a nearby or remote cluster, or they could be geographically dispersed computers that are available under one of the increasingly many modalities of Internet-based computing—such as Grid computing (cf. [5]), or volunteer computing (cf. [9]). In order to avoid unintended connotations concerning the organization of the remote computers, we avoid evocative terms in favor of the generic "assemblage." Advances in computing power never come without cost. These new platforms add various types of *uncertainty* to the list of concerns that must be addressed when preparing one's computation for allocation to the available computers: notably, computers can slow down unexpectedly, even failing ever to complete allocated work. The current paper follows in the steps of sources such as [1,4,11], analytic studies of algorithmic techniques for coping with uncertainty in computational settings. Whereas most of these sources address the uncertainty of the computers in an assemblage one computer at a time, we attempt here to view the assemblage as a "team" wherein one computer's shortcomings can be compensated for by other computers, most notably by judiciously *replicating work*, i.e., allocating some work to more than one computer.

The problem. We have a large computational workload whose constituent work is *divisible* in the sense that each chunk of work can be partitioned into arbitrary granularity (cf. [7]). We also have access to $p \geq 1$ identical computers to help us compute the workload via *worksharing*. We study homogeneous assemblages in the current paper in order to concentrate only on developing technical tools to cope with uncertainty within an assemblage. We hope to focus in later work on the added complexity of coping with uncertainty within a *heterogeneous* assemblage.

We address here the most draconian type of uncertainty that can plague an assemblage of computers, namely, vulnerability to *unrecoverable interruptions* that cause us to lose all work currently in progress on the interrupted computer. We wish to cope with such interruptions—whether they arise from hardware failures or from a loaned/rented computer's being reclaimed by its owner, as during an episode of *cycle-stealing* (cf. [1, 4, 11]). Our scheduling tool is *work replication*, the allocation of work to more than one remote computer. The only external resource to help us use this tool judiciously is our assumed access to *a priori* knowledge of the risk of a computer's having been interrupted—which we assume is the same for all computers (as in [4, 11], our

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

scheduling strategies can be adapted to use statistical, rather than exact, knowledge of the risk of interruption—albeit at the cost of weakened performance guarantees).

The goal. Our goal is to maximize the *expected amount of work that gets computed by the assemblage of computers*, no matter which, or how many computers get interrupted. Therefore, we implicitly assume that we are dealing with applications for which even partial output is meaningful, e.g., annotation of metagenomics data. In metagenomics annotation, one has a large number of DNA fragments to classify (as belonging to eukaryotes, prokaryotes, etc.); one would rather have all the DNA fragments processed, but the result of the classification is nevertheless meaningful even if the annotation is fragmentary (this just artificially raises the "unknown" category).

The challenges. The challenges of scheduling our workload on interruptible remote computers can be described in terms of dilemmas. Sending each remote computer a small amount of work minimizes vulnerability to interruption-induced losses, but it maximizes the impact of per-work overhead and minimizes the opportunities for "parallelism" within the assemblage of remote computers. *Replicating work* lessens our vulnerability to interruption-induced losses, but it minimizes the expected productivity advantage from having access to remote computers. (The pros and cons of work replication are discussed in [8].)

Approaches to the challenges. *(1) "Chunking" our workload.* We cope with the first dilemma by sending work allocations to computers as a sequence of *chunks* rather than as a single block per computer. (We use the generic "chunk" instead of "task" to emphasize tasks' divisibility.) This approach, advocated in [4, 11], allows each computer to *checkpoint* at various times, thereby, protecting some work from the threat of interruption. *(2) Replicating work.* We allocate some chunks to more than one remote computer in order to enhance their chances of being computed successfully. We replicate work judiciously, in deference to the second dilemma.

Because communication to remote computers is likely to be costly in time and overhead, we limit such communication by orchestrating work replications in an *a priori,* static manner, rather than dynamically, in response to observed interruptions. While we thereby duplicate work unnecessarily when there are few interruptions among the remote computers, we also prevent the server from becoming a communication bottleneck.

Main results. Most of our results assume the *linear risk* model, in which the probability that a computer will be interrupted increases linearly with the time the computer has been available. We find optimal schedules for 1 remote computer, both when there are per-chunk overheads and when there are not (Section 3). We find asymptotically optimal schedules when there are 2 remote computers (Section 4). We propose heuristics for the general p-computer case (Section 5), whose efficiency is established through extensive simulations (Section 6).

Before embarking on technicalities, we quickly discuss related work. *Due to space limitations, we relegate all proofs and more discussion of the related work to the companion research report* [3].

Related work. The literature contains relatively few rigorously analyzed scheduling algorithms for interruptible "parallel" computing in assemblages of computers. Among those we know of, [1, 4, 11] deal with an *adversarial* model of interruptible computing. One finds in [1] a randomized scheduling strategy which, with high probability, completes within a logarithmic factor of the optimal fraction of the workload. In [4, 11], scheduling is a game against a malicious adversary who seeks to minimize work production by interrupting each remote computer in order to kill all work in progress. Among experimental sources, [13] studies the use of task replication on a heterogeneous desktop grid whose constituent computers may become definitively unavailable; the objective is to eventually process all work.

There is a large literature on scheduling divisible workloads on assemblages of computers that are not vulnerable to interruptions. We refer the reader to [7] and its myriad intellectual progeny; another good start is [2], a thorough study of divisible load scheduling on star and tree networks; also, [6] shows how a linear model, like our free-initiation model, can lead to absurd schedules involving infinitely many infinitely small chunks. We encounter such a problem in Section 3.1.

We do not enumerate here the many studies of computation on assemblages of remote computers, which focus either on systems that enable such computation or on specific algorithmic applications. However, we point to [10] and [12] as exemplars of the two types of studies.

2 The Technical Framework

We supply the technical details necessary to turn the informal discussion in the Introduction into a framework in which we can develop and rigorously validate scheduling guidelines.

2.1 The Computation and the Computers

We have W units of divisible work to execute on an assemblage of $p \geq 1$ identical computers that are susceptible to unrecoverable interruptions that "kill" all work in progress. All computers share the same perfectly known instantaneous probability of being interrupted, and this probability increases with the amount of time the computer has been operating (whether working or not). As discussed in the Introduction, the danger of losing work in progress when an interruption incurs mandates that we not just divide our workload into W/p equal-size chunks and allocate one chunk to each computer. Instead, we "protect" our workload as best we can, by:

- partitioning it into chunks, the unit of work that we allocate to the computers;
- prescribing a schedule for allocating chunks to computers;
- allocating some chunks to several computers, as a divisible-load analogue of task replication.

We also try to avoid having *our* computer become a communication bottleneck, by orchestrating chunk replications in an *a priori,* static manner—even though this leads to duplicated work when there are few or no interruptions—rather than dynamically, in response to observed interruptions.

2.2 Modeling Interruptions and Expected Work

2.2.1 The interruption model

Within our model, all computers share the same *risk function*, i.e., the same instantaneous probability, $Pr(w)$, of having been interrupted by the end of "the first w time units." We measure time in terms of work units that *could have been* executed "successfully," i.e., with no interruption. In other words "the first w time units" is the amount of time that a computer *would have needed* to compute w work units *if* it had started working on them when the entire worksharing episode began. This time scale is shared by all computers. Of course, $Pr(w)$ increases with w; we assume that we know its exact value (see the comment in the Introduction).

It is useful in our study to generalize our measure of risk by allowing one to consider many baseline moments. We denote by $Pr(s, w)$ the probability that a computer has not been interrupted during the first s "time units" but has been interrupted by "time" $s + w$. Thus, $Pr(w) = Pr(0, w)$ and $Pr(s, w) = Pr(s + w) - Pr(s)$. We let $\kappa \in (0, 1]$ be a constant that weights our probabilities. We illustrate the role of κ as we introduce our risk function.

Linearly increasing risk. The risk function that is the focus of our study is the linear function $Pr(w) = \kappa w$. It is the most natural model in the absence of further information: the risk of interruption grows linearly with the time that the computer has been available, or equivalently to the amount of work it could have done. The density function is then $dPr = \kappa dt$ for $t \in [0, 1/\kappa]$ and 0 otherwise, so that

$$Pr(s, w) = \min\left\{1, \int_s^{s+w} \kappa dt\right\} = \min\{1, \kappa w\}.$$

The constant $1/\kappa$ recurs repeatedly in our analyses, since it can be viewed as the time by which an interruption will have occurred with probability 1. To enhance legibility of the rather complicated expressions that populate our analyses, we denote $1/\kappa$ by X.

2.2.2 Expected work production

Risk functions help us finding an efficient way to chunk work for, and allocate work to, the remote computers, in order to maximize the expected work production of the assemblage. Let jobdone be the random variable whose value is the number of work units that the assemblage executes successfully under a given scheduling regimen. Formally, we are striving to maximize the expected value (or, expectation) of jobdone.

We perform our study under two models, which play different roles as one contemplates the problem of scheduling a large workload. The models differ in the way chunk execution times relate to chunk sizes. In short, there are two classes of time-costs, those that are proportional to the chunk size and those that are fixed constants. When chunks are large, the second cost will be minuscule compared to the first. This suggests that the fixed costs can be ignored, but one must be careful: if one ignores the fixed costs, then there is no disincentive to, say, deploying the workload to the remote computers in $n + 1$ chunks, rather than n. Of course, increasing the number of chunks tends to make chunks smaller—which increases the significance of the fixed cost! Therefore, we perform the current study with two cost models, striving for optimal schedules under each one. (1) The *free-initiation model* is characterized by *not* charging the owner of the workload a per-chunk fixed cost. This model focuses on situations wherein the fixed costs are negligible compared to the chunk-size-dependent costs. (2) The *charged-initiation model,* which more accurately reflects the costs incurred with real computing systems, is characterized by accounting for both the fixed and chunk-dependent costs.

The *free-initiation* model. This model assesses no per-chunk cost. Our results under this model approximate reality well when *a priori* chunks must be large, e.g., because large fixed-size costs demand that every remote computer do a substantial amount of work in order to amortize the fixed-size costs. In such a situation, one keeps chunks large by placing a bound on the number of scheduling "rounds," which counteracts this model's tendency to increase the number of "rounds" without bound. Importantly also: the free-initiation model allows us to obtain *bounds* on the expectation of jobdone under the charged-initiation model, when such bounds are prohibitively hard to derive directly; cf. Theorem 1.

Under the free-initiation model, the expected value of jobdone under a given scheduling regimen Θ, denoted $E^{(\mathrm{f})}(\Theta)$, the superscript "f" denoting "free(-initiation)," is

$$E^{(\mathrm{f})}(\Theta) \;=\; \int_0^\infty Pr(\mathsf{jobdone} \geq u \text{ under } \Theta)\, du.$$

We illustrate this model via three calculations of $E^{(\mathrm{f})}(\Theta)$ in cases when Θ deploys the whole workload on a single computer. To enhance legibility, let the phrase "under Θ" within "$Pr(\mathsf{jobdone} \geq u \text{ under } \Theta)$" be specified implicitly by context. For an arbitrary risk function Pr:

Workload deployed as 1 chunk (regimen Θ_1)

$$E^{(\mathrm{f})}(W,\Theta_1) = \; W\,(1 - Pr(W))$$

Workload deployed as 2 chunks (regimen Θ_2)

$$\begin{aligned} E^{(\mathrm{f})}&(W,\Theta_2) \\ &= \int_0^{\omega_1} Pr(\mathsf{jobdone} \geq u)du \\ &\quad + \int_{\omega_1}^{\omega_1+\omega_2} Pr(\mathsf{jobdone} \geq u)du \\ &= \omega_1(1 - Pr(\omega_1)) \;+\; \omega_2(1 - Pr(\omega_1+\omega_2)) \end{aligned}$$

Workload deployed as n chunks (regimen Θ_n)

$$\begin{array}{rl} \text{Chunk sizes:} & \omega_1 \geq 0, \omega_2 \geq 0, \ldots, \omega_n \geq 0 \\ \text{where} & \omega_1 + \cdots + \omega_n = W \\ E^{(\mathrm{f})}(W,\Theta_n) = & \omega_1(1 - Pr(\omega_1)) \;+ \cdots + \\ & + \cdots + \; \omega_n(1 - Pr(\omega_1 + \cdots + \omega_n)). \end{array}$$

Optimizing expected work-production on one remote computer. One goal of our study is to learn how to craft, for each integer n, a scheduling regimen Θ that maximizes $E^{(\mathrm{f})}(W,\Theta)$. We observe that many risk functions—such as the linear risk function—represent situations wherein the remote computers are *certain* to have been interrupted no later than a known eventual time. In such a situation, one might get more work done, in expectation, by not deploying the entire workload: one could increase the expectation by making the last deployed chunk even a tiny bit smaller than needed to deploy all W units of work (this will be the case in Theorem 2 for the free-initiation model and in Theorem 3 for the charged-initiation model). Thus, our ultimate goal when considering a single remote computer (the case $p = 1$), is to determine, for each integer n:

- how to select n chunk sizes that collectively sum to *at most* W,
- how to select n chunks of these sizes out of our workload,
- how to schedule the deployment of these chunks

in a way that maximizes the expected amount of work that gets done. We formalize this goal via the function $E^{(\mathrm{f})}(W,n) \;=\; \max\{\omega_1(1 - Pr(\omega_1)) + \cdots + \omega_n(1 - Pr(\omega_1 + \cdots + \omega_n))\}$, where the maximization is over all n-tuples $\{\omega_1 \geq 0,\ \omega_2 \geq 0, \ldots,\ \omega_n \geq 0\}$ such that $\omega_1 + \omega_2 + \cdots + \omega_n \leq W$.

The *charged-initiation* model. This model is much harder to analyze than the free-initiation model, even when there is only one remote computer. In compensation, this model often allows one to determine analytically what the best numbers of chunks are. Under this model, the overhead for each additional chunk is a fixed cost—which, in common with time, we measure in units of work—that is added to the cost of computing of each chunk; we denote this overhead by ε (for instance this may correspond to a checkpointing cost). Under this model, then, the expectation of jobdone under schedule Θ, denoted $E^{(\mathrm{c})}(\Theta)$, the superscript "c" denoting "charged(-initiation)," is

$$E^{(\mathrm{c})}(\Theta) \;=\; \int_0^\infty Pr(\mathsf{jobdone} \geq u + \varepsilon)\, du.$$

Letting $E^{(\mathrm{c})}(W,k)$ be the analogue for the charged-initiation model of the parameterized free-initiation expectation $E^{(\mathrm{f})}(W,k)$, we find that, when the whole workload is deployed as a single chunk, $E^{(\mathrm{c})}(W,\Theta_1) = W\,(1 - Pr(W+\varepsilon))$, and when work is deployed as two chunks of respective sizes ω_1 and ω_2, $E^{(\mathrm{c})}(W,\Theta_2) = \omega_1(1 - Pr(\omega_1+\varepsilon)) + \omega_2(1 - Pr(\omega_1+\omega_2+2\varepsilon))$.

Relating the two models. One can bound the work completed under the charged-initiation model via the free-initiation model. This justifies our primary focus on the free-initiation model.

Theorem 1 *Let $E^{(\mathrm{c})}(W,n)$ and $E^{(\mathrm{f})}(W,n)$ denote, respectively, the optimal n-chunk expected value of* jobdone *under the* charged-*initiation model and under the* free-*initiation model. Then:*

$$E^{(\mathrm{f})}(W,n) \;\geq\; E^{(\mathrm{c})}(W,n) \;\geq\; E^{(\mathrm{f})}(W,n) - n\varepsilon.$$

3 Scheduling for a Single Remote Computer

This section is devoted to studying how to schedule optimally when there is only a single remote computer, under the linear risk model. Some of the results we derive bear a striking similarity to their analogues in [4], despite certain substantive differences in models.

3.1 An Optimal Schedule under the Free-Initiation Model

We first illustrate why the risk of losing work because of an interruption must affect our scheduling strategy, even when there is only one remote computer. When $W \leq 1/\kappa$, the expected amount of work achieved when one deploys the entire workload in a single chunk is $E^{(\mathrm{f})}(W, \Theta_1) = W - \kappa W^2$. The analogous quantity when one deploys the workload in two chunks, of respective sizes $\omega_1 > 0$ and $\omega_2 > 0$, with $\omega_1 + \omega_2 = W$, is $E^{(\mathrm{f})}(W, \Theta_2) = W - W^2\kappa + \omega_1\omega_2\kappa$. Note that: $E^{(\mathrm{f})}(W, \Theta_2) - E^{(\mathrm{f})}(W, \Theta_1) = \omega_1\omega_2\kappa > 0$. Thus, as one would expect: *For any fixed total workload, one increases the expectation of* jobdone *by deploying the workload as two chunks, rather than one—no matter how one sizes the chunks.* In fact, the expectation of jobdone for the optimal schedule strictly increases with the number of chunks allowed (Theorem 2). This fact identifies a weakness of the free-initiation model: increasing the number of chunks always increases the expected amount of work done—so the (unachievable) "optimal" strategy would deploy infinitely many infinitely small chunks.

Theorem 2 *Say that one wishes to deploy W units of work to a single remote computer in at most n chunks, for some integer $n > 0$. In order to maximize the expectation of* jobdone*, one should have all chunks comprise Z/n units of work, where $Z = \min\left\{W, \frac{n}{n+1}X\right\}$. In expectation, this optimal schedule completes $E^{(\mathrm{f})}(W, n) = Z - \frac{n+1}{2n}Z^2\kappa$ units of work.*

3.2 An Optimal Schedule under the Charged-Initiation Model

The *charged-initiation* analogue of Theorem 2 is dramatically more difficult to deal with.

Theorem 3 *Say that one wishes to deploy W units of work to a single remote computer in at most n chunks, for some integer $n > 0$; say that $X \geq \varepsilon$. Let $n_1 = \left\lfloor \frac{1}{2}\left(\sqrt{1 + 8X/\varepsilon} - 1\right)\right\rfloor$ and $n_2 = \left\lfloor \frac{1}{2}\left(\sqrt{1 + 8W/\varepsilon} + 1\right)\right\rfloor$. The unique regimen for maximizing $E^{(\mathrm{c})}(W, n)$ specifies $m = \min\{n, n_1, n_2\}$ chunks: the first has size $\omega_{1,m} = \frac{Z}{m} + \frac{m-1}{2}\varepsilon$ where $Z = \min\left\{W, \frac{m}{m+1}X - \frac{m}{2}\varepsilon\right\}$; the $(i+1)$th chunk inductively has size $\omega_{i+1,m} = \omega_{i,m} - \varepsilon$. In expectation, this schedule completes*

$$E^{(\mathrm{c})}(W, n) = Z - \frac{m+1}{2m}Z^2\kappa - \frac{m+1}{2}Z\varepsilon\kappa + \frac{(m-1)m(m+1)}{24}\varepsilon^2\kappa$$

units of work.

4 Scheduling for Two Remote Computers

The case of *two* remote computers affords us useful principles for addressing the general case of p remote computers. We first establish characteristics of optimal schedules under general risk functions, then restrict attention to linear risks. Throughout this section, we consider two remote computers, P_1 and P_2, under the *free-initiation* model.

4.1 Two Remote Computers under General Risk

Say, for $i = 1, 2$, that we deploy n_i chunks of work, $\mathcal{W}_{i,1}$, ..., $\mathcal{W}_{i,n_i}$, on P_i; P_i must schedule them in this order. We do not assume any *a priori* relation between how P_1 and P_2 break their allocated work into chunks: work that is allocated to both P_1 and P_2 may be chunked differently on the two machines.

Theorem 4 *Let Θ be a schedule for P_1 and P_2. Say that, for both computers, the probability of being interrupted never decreases as a computer processes more work. Then there exists a schedule Θ' that, in expectation, completes as much work as does Θ and that satisfies the following three properties; cf. Fig. 1.*

1. Maximal work deployment. *Θ' deploys as much of the overall workload as possible: the workloads it deploys to P_1 and P_2 can overlap only if their union is the entire overall workload.*

2. Local work priority. *Θ' has P_1 (resp., P_2) process all of the allocated work that it does* not *share with P_2 (resp., P_1) before it processes any shared work.*

3. Shared work "mirroring." *Θ' has P_1 and P_2 process their shared work "in opposite orders." Specifically, say for $k = 1, 2$, that P_k chops its allocated work into chunks $\mathcal{W}_{k,1}, \ldots, \mathcal{W}_{k,n_k}$. Say that there exist chunk-indices a_1, $b_1 > a_1$ for P_1, and a_2, $b_2 > a_2$ for P_2 such that: $\mathcal{W}_{1,a_1}$ and $\mathcal{W}_{2,a_2}$ both contain a shared "piece of work" A, and $\mathcal{W}_{1,b_1}$ and $\mathcal{W}_{2,b_2}$ both contain a shared "piece of work" B. Then if Θ' has P_1 execute A before B (i.e., P_1 executes $\mathcal{W}_{1,a_1}$ before $\mathcal{W}_{1,b_1}$), then Θ' has P_2 execute B before A (i.e., P_2 executes $\mathcal{W}_{2,b_2}$ before $\mathcal{W}_{2,a_2}$).*

4.2 Two Remote Computers under Linear Risk

Optimal schedules. We now specialize from general risk functions to linear ones. Results in [3] suggest that finding

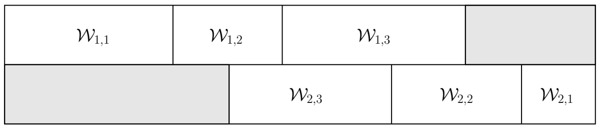


Figure 1. The shape of Theorem 4's optimal schedule for two computers; $n_1 = n_2 = 3$. The top row displays P_1's chunks, the bottom row P_2's. Vertically aligned parts of chunks correspond to shared work; shaded areas depict unallocated work (e.g., no work in $\mathcal{W}_{2,1}$ is allocated to P_1).

exactly optimal schedules is surprisingly difficult, even in the simple case wherein each computer processes its allocated work as a single chunk. Nevertheless, it is worth seeking significant restricted situations wherein one can tractably discover exactly optimal schedules. One obvious candidate for special consideration is the family of schedules that allocate the entire workload to each remote computer—which seems to be desirable when $W\kappa$ is small enough. We conjecture that, for such schedules, an optimal strategy would have the two computers chop the workload into chunks of the same size and then process these chunks in "opposite orders" (as defined in the third property of Theorem 4). When all remote computers chop the workload into n chunks, this scheduling strategy completes, in expectation, $W - \frac{W^3\kappa^2}{6}\left(1 + \frac{3}{n} + \frac{2}{n^2}\right)$ units of work (cf. Theorem 5 below). Extensive numerical simulations suggest that such a scheduling strategy is, indeed, optimal as long as $n \leq 3$. However, we know that the strategy is *suboptimal* once one allows n to exceed 3. Indeed, for $n = 4$, the strategy completes, in expectation, $W - \frac{5}{16}W^3\kappa^2$ units of work, which is strictly less than the $W - \frac{757-73\sqrt{73}}{432}W^3\kappa^2$ units completed, in expectation, by the strategy specified schematically in Fig. 2 (with $m = 1$ and $\alpha = \frac{\sqrt{73}-7}{6}W$; the boxes in Fig. 2 contain chunk sizes). Furthermore, the schedule in Fig. 2 is suboptimal as soon as we allow computers to chop work into eight chunks. To wit, Fig. 3 presents an 8-chunk schedule that completes, in expectation, $W - \frac{229-44\sqrt{22}}{98}W^3\kappa^2 \approx W - 0.230834W^3\kappa^2$ units of work, when $\alpha = \frac{4\sqrt{22}-17}{14}W$, while the schedule of Fig. 2, using 8 chunks per computer (specifically, $m = 5$ and $\alpha = \frac{19-\sqrt{193}}{42}W$) completes, in expectation, $W - \frac{18293-965\sqrt{193}}{21168}W^3\kappa^2 \approx W - 0.230857W^3\kappa^2$ units of work. (The schedule of Fig. 3 is not even optimal for 8 chunks, but the best schedule we found numerically was almost identical but slightly less regular.)

The increasing complexities of the preceding "counterexample" schedules suggest how hard it will be to search for, and characterize, exactly optimal schedules—even in the presence of simplifying assumptions, such as that the whole workload is distributed to each computer. Since our simulations suggest that simple regular solutions often complete, in expectation, almost as much work as do complex exactly optimal schedules, we henceforth aim for simply structured asymptotically optimal schedules.

Asymptotically optimal schedules. We now analyze Algorithm 1, whose prescribed schedules for two remote computers are asymptotically optimal; they are exactly optimal when $W\kappa \geq 2$.

Notation. Given a workload of size W that is ordered linearly, $\langle a, b\rangle$ denotes the sub-workload obtained by eliminating the initial a units of work and all work beyond the initial b units. Thus, $\langle 0, W\rangle$ denotes the entire workload, $\langle 0, \frac{1}{2}W\rangle$ its first half, and $\langle \frac{1}{2}W, W\rangle$ its second half.

Algorithm 1: Scheduling for 2 computers using at most n chunks per computer

```
1  if $W \geq 2X$ then
2      $\forall i \in [1, n],\ \mathcal{W}_{1,i} \leftarrow \left\langle \frac{i-1}{n} \cdot \frac{n}{n+1}X,\ \frac{i}{n} \cdot \frac{n}{n+1}X \right\rangle$
3      $\forall i \in [1, n]$,
4          $\mathcal{W}_{2,i} \leftarrow \left\langle W - \frac{i}{n} \cdot \frac{n}{n+1}X,\ W - \frac{i-1}{n} \cdot \frac{n}{n+1}X \right\rangle$
5  if $W \leq X$ then
6      $\forall i \in [1, n], \mathcal{W}_{1,i} = \mathcal{W}_{2,n-i+1} \leftarrow \left\langle \frac{i-1}{n}W,\ \frac{i}{n}W \right\rangle$
7  if $X < W < 2X$ then
8      $\ell \leftarrow \lfloor n/3 \rfloor$
9      $\forall i \in [1, \ell],\ \mathcal{W}_{1,i} \leftarrow \langle \frac{i-1}{\ell}(W - X),\ \frac{i}{\ell}(W - X)\rangle$
10     $\forall i \in [1, \ell]$,
11         $\mathcal{W}_{2,i} \leftarrow \langle W - \frac{i}{\ell}(W - X),\ W - \frac{i-1}{\ell}(W - X)\rangle$
12     $\forall i \in [1, 2l], \mathcal{W}_{1,l+i} = \mathcal{W}_{2,3l-i+1} \leftarrow$
           $\langle (W-X) + \frac{i-1}{2\ell}(2X-W), (W-X)+\frac{i}{2\ell}(2X-W)\rangle$
```

Theorem 5 *The schedules specified by Algorithm 1 are:*

1. *optimal when $W \geq 2X$; in expectation, they complete $\frac{n-1}{n}X$ units of work, which tends to X ("tends to" means "as n grows without bound.");*
2. *asymptotically optimal when $W \leq X$; in expectation, they complete $W - \frac{1}{6}W^3\kappa^2\left(1 + \frac{3}{n} + \frac{2}{n^2}\right)$ units of work, which tends to $W - \frac{1}{6}W^3\kappa^2$;*
3. *asymptotically optimal when $X < W < 2X$; letting $\ell = \lfloor n/3 \rfloor$, in expectation, they complete $2W - \frac{1}{3}X - W^2\kappa + \frac{1}{6}W^3\kappa^2 + \frac{1}{\ell}\left(\left(1 + \frac{1}{\ell}\right)W - \left(1 + \frac{2}{3\ell}\right)X - \frac{1}{2\ell}W^2\kappa - \frac{1}{4}\left(1 - \frac{1}{3\ell}\right)W^3\kappa^2\right)$ units of work, which tends to $2W - \frac{1}{3}X - W^2\kappa + \frac{1}{6}W^3\kappa^2$.*

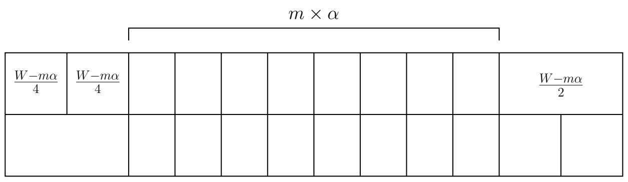


Figure 2. Counterexample to the optimality of schedules that employ equal-size chunks.

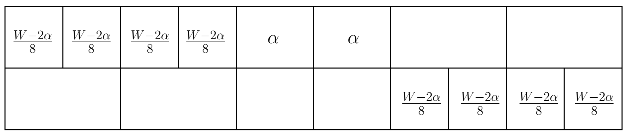


Figure 3. Counterexample to the optimality of the schedule of Fig. 2.

5 Scheduling for p Remote Computers

We turn finally to the general case of p remote computers. This case is so much more difficult than the already challenging case $p = 2$, that we devote our efforts here to searching for *efficient heuristic schedules*. We adopt a pragmatic approach by restricting attention to "well-structured" schedules that employ same-size chunks; this restriction has two major antecedents. (1) The optimal schedules for the case $p = 1$ and the asymptotically schedules for the case $p = 2$ mandate using same-size chunks; (2) This restriction greatly simplifies the specification and implementation of schedules, by imposing simplifying structure on the scheduling problem.

All of the schedules we develop here operate as follows: (1) they *partition* the total workload into (disjoint) *slices* that they assign to—and replicate on—disjoint subsets (*coteries*) of remote computers; each coterie partitions each slice into *same-size* chunks; (2) they orchestrate the processing of the slices on each coterie of remote computers.

5.1 The Partitioning Phase

We begin with some simple partitioning heuristics; these are tailored to the linear risk function but can be adapted easily to other risk functions. We break the scheduling problem into three subproblems, based on the size of our workload. Our partition acknowledges the futility of deploying a workslice of size $> X$ on any computer, in the light of our interruption model.

- W **is "very small."** When $W \leq X$, we deploy the entire workload in a single slice, that is replicated on all p computers.
- W **is "very large."** When $W \geq pX$, we deploy p slices of common size X, to be processed independently on the remote computers. We abandon the remaining $W - pX$ units of work, in acknowledgment of our interruption model.
- W **is of "intermediate" size.** The case $X < W < pX$ is the interesting challenge, as there is no compelling scheduling strategy. In this case, we deploy W units of work on the p remote computers. We partition this work into $q = \lceil Z\kappa \rceil$ slices, each of size $\mathsf{sl} = Z/q$, then deploy these slices on disjoint coteries of remote computers. We load balance computing resources as much as possible, by replicating each slice on either $\lfloor p/q \rfloor$ or $\lceil p/q \rceil$ remote computers.

For general risk functions, we introduce a parameter λ that specifies the maximum probability of a slice's being interrupted that the user is willing to risk. We use λ to compute the maximum allowable slice size maxsl by insisting that $Pr(\mathsf{maxsl}) = \lambda$. For instance, if $\lambda = 1/2$, then under the linear risk function, we set $\mathsf{maxsl} = \frac{1}{2}X$, while with an exponential risk function we set $\mathsf{maxsl} = (\ln 2)X$. The amount of work we actually deploy is $Z = \min(W, p \times \mathsf{maxsl})$. This mandates using $q = \lceil Z/\mathsf{maxsl} \rceil$ slices, of size $\mathsf{sl} = Z/q$.

5.2 The Orchestration Phase

The partition phase has left us with independent slices of work, each of size $\mathsf{sl} \leq \mathsf{maxsl}$, to be processed by disjoint coteries of computers. All slices are partitioned into n chunks of common size $\omega = \mathsf{sl}/\mathsf{n}$. For each coterie Γ, each chunk assigned to Γ will be processed by all g_Γ computers in the coterie. Our challenge is to determine when (at which time step) and where (on which computer) to execute which chunk, in a way that maximizes the expected amount of work completed by the total assemblage of p computers.

5.2.1 General schedules

Let us motivate our approach to the orchestration problem via the following example, wherein each slice is partitioned into $\mathsf{n} = 12$ chunks, and each coterie contains $\mathsf{g} = 4$ computers. Since each coterie operates independently of all others, we only need specify the schedule for a single coterie. For a coterie Γ and its associated slice, we represent a possible schedule via an *execution chart* such as that of Table 1. Rows of this chart enumerate the computers in Γ, and columns enumerate the indices of the chunks into which Γ's slice is chopped. Chart-entry $C_{i,j}$ is the step at which chunk j is processed by computer P_i.

Any $\mathsf{g} \times \mathsf{n}$ integer matrix whose rows are permutations of $[1..\mathsf{n}]$ is a valid execution chart, under which each P_i executes once each chunk j (at step $C_{i,j}$). One can use such a chart to calculate the expected amount of work completed under the schedule that the chart specifies. To wit, chunk j will *not* be executed under the schedule only if all g computers in the coterie are interrupted before they complete the chunk. This occurs with probability $\prod_{i=1}^{\mathsf{g}} Pr(C_{i,j}\omega) = \prod_{i=1}^{\mathsf{g}} Pr\left(C_{i,j}\mathsf{sl}/\mathsf{n}\right)$, so the expectation of the total work completed from the slice is

$$E(\mathsf{sl},\mathsf{n}) \;=\; \mathsf{sl}\left(1 - \frac{1}{\mathsf{n}}\left(\frac{\mathsf{sl}\kappa}{\mathsf{n}}\right)^{\mathsf{g}} \sum_{j=1}^{\mathsf{n}}\prod_{i=1}^{\mathsf{g}} C_{i,j}\right).$$

The last expression is specific to the linear risk function: because each $C_{i,j} \leq n$, we have $Pr(C_{i,j}\omega) = C_{i,j}\omega\kappa$ under this risk function, so we need not take the minimum of the last expression with 1. We can, therefore, derive the following upper bound:

Proposition 1

$$E(\mathsf{sl},\mathsf{n}) \leq \mathsf{sl}\cdot\left(1 - \left(\mathsf{sl}\cdot\kappa\frac{(\mathsf{n}!)^{1/\mathsf{n}}}{\mathsf{n}}\right)^{\mathsf{g}}\right) \approx \mathsf{sl}\cdot\left(1 - \left(\frac{\mathsf{sl}.\kappa}{e}\right)^{\mathsf{g}}\right).$$

5.2.2 Group schedules: introduction

Referring back to Table 1, we note that chunks 1–4 are always executed at the same steps; the same is true for chunks 5–8 as a group and chunks 9–12 as a group. The slice's chunks thus partition naturally into three *groups*. By respecifying the schedule of Table 1 as the *group(-oriented)* schedule of Table 2, we significantly simplify the specification. In the group(-oriented) execution chart of Table 2, each column corresponds to a group of chunks, and the ith row specifies the step at which chunks are executed for the ith time. It is implicit that chunk-indices within each group are cyclically permuted (downward) at each step, so that each chunk is processed by each computer exactly once.

We generalize this description. When n is a multiple of g, we can sometimes convert the full $\mathsf{g} \times \mathsf{n}$ execution chart C exemplified by Table 1 to the $\mathsf{g} \times \mathsf{n}/\mathsf{g}$ *group(-oriented)* execution chart $\widehat{C}$ exemplified by Table 2. There are n/g groups of size g, and chart-entry $\widehat{C}_{i,j}$ denotes the step at which group j of chunks is executed for the ith time. For a chart $\widehat{C}$ to specify a valid group schedule, its total set of entries must be a permutation of $[1..n]$. When $\widehat{C}$ does specify a valid group schedule, the expected amount of work it completes, under the linear risk model, is:

$$E(\mathsf{sl},\mathsf{n},\Theta) \;=\; \mathsf{sl} - \mathsf{K}^{(\Theta)}\cdot\frac{\mathsf{g}}{\kappa}\left(\frac{\mathsf{sl}\kappa}{\mathsf{n}}\right)^{\mathsf{g}+1}$$
$$\text{where } \mathsf{K}^{(\Theta)} \;=\; \sum_{j=1}^{\mathsf{n}/\mathsf{g}}\prod_{i=1}^{\mathsf{g}} \widehat{C}_{i,j}^{(\Theta)}.$$

Thus: *A smaller value of* $\mathsf{K}^{(\Theta)}$ *corresponds to a larger value of* $E(\mathsf{sl},\mathsf{n},\Theta)$.

Group schedules are very natural, because they are *symmetric:* all computers play the same role as the work is processed, differing only in the times at which they process different chunks. Intuition suggests that the most productive schedules are symmetric: why should some of the identical computers be treated differently by "nature" than others? Indeed, the following upper bound on the expected work production of group schedules does not distinguish symmetric schedules from general ones (but we have not yet been able to prove that no difference exists).

Proposition 2 *For any group schedule* Θ,

$$E(\mathsf{sl},\mathsf{n},\Theta) \;\leq\; \mathsf{sl}\cdot\left(1 - \left(\frac{\mathsf{sl}\kappa(\mathsf{n}!)^{1/\mathsf{n}}}{\mathsf{n}}\right)^{\mathsf{g}}\right).$$

Proposition 2 affords us an easy lower bound on the K value of any group schedule with parameters g and n:
$\mathsf{K}_{\min} \;=\; \left\lceil(\mathsf{n}/\mathsf{g})(\mathsf{n}!)^{\mathsf{g}/\mathsf{n}}\right\rceil$.

5.2.3 Group schedules: specific schedules

Our group schedules strive to maximize expected work completion by having every computer attempt to compute every chunk. Of course, there are many ways to achieve this coverage, and the form of the risk function will make some ways more advantageous than others with respect to maximizing expected work completion. As an extreme example, when $p = 2$ it is *always* advantageous to have the remote computers process the work they share "in opposite orders" (Theorem 4). We now specify and compare the performance of six group schedules whose chunk-scheduling regimens seem to be a good match for the way the linear risk function "predicts" interruptions.

Cyclic scheduling (Table 3(a)). Under this regimen, Θ_{cyclic}, groups are executed sequentially, in a round-robin fashion, so the chunks of group j are executed at steps j, $j + \mathsf{n}/\mathsf{g}$, $j + 2\mathsf{n}/\mathsf{g}$, and so on. The weakness of Θ_{cyclic} is that chunks in low-index groups are more likely to be completed than are chunks in high-index groups—because

Computer \ Chunk	1	2	3	4	5	6	7	8	9	10	11	12
P_1	1	6	9	12	2	5	8	11	3	4	7	10
P_2	12	1	6	9	11	2	5	8	10	3	4	7
P_3	9	12	1	6	8	11	2	5	7	10	3	4
P_4	6	9	12	1	5	8	11	2	4	7	10	3

Table 1. General schedule.

Group 1	Group 2	Group 3
chunks 1–4	chunks 5–8	chunks 9–12
1	2	3
6	5	4
9	8	7
12	11	10

Table 2. Group-oriented schedule.

chunks remain in the same relative order throughout the computation. The remaining schedules aim to compensate for this imbalance.

Reverse scheduling (Table 3(b)). A schedule $\Theta_{\mathsf{reverse}}$ produced under this regimen executes the chunks in each group once in the initially-specified order, and then executes them in the *reverse* order $\mathsf{n}/\mathsf{g} - 1$ times, so the chunks in group j are executed at step j, and thereafter at steps $2\mathsf{n}/\mathsf{g} - j + 1$, $3\mathsf{n}/\mathsf{g} - j + 1$, $4\mathsf{n}/\mathsf{g} - j + 1$, and so on. $\Theta_{\mathsf{reverse}}$ thereby strives to compensate for the imbalance in chunks' likelihoods of being completed created by their initial order of processing. ($\Theta_{\mathsf{reverse}}$ is the schedule specified in Table 2.)

Mirror scheduling (Table 3(c)). The mirror schedule Θ_{mirror}, which is defined for even g, is a compromise between the cyclic and reverse strategies. Θ_{mirror} compensates for the imbalance in likelihood of completion only during the second half of the computation, by mimicking Θ_{cyclic} for the first $\mathsf{g}/2$ phases of processing a group and mimicking $\Theta_{\mathsf{reverse}}$ thereafter.

Snake-like scheduling (Table 3(d)). Θ_{snake} compensates for the imbalance of cyclic scheduling by mimicking Θ_{cyclic} and $\Theta_{\mathsf{reverse}}$ at alternating steps, which lends a snake-like structure to the execution chart $\widehat{C}^{(\Theta_{\mathsf{snake}})}$.

Fat snake-like scheduling (Table 3(e)). $\Theta_{\mathsf{fat-snake}}$ qualitatively adopts the same strategy as Θ_{snake}, but it slows down the return phase. Consider three consecutive rows of $\widehat{C}^{(\Theta_{\mathsf{fat-snake}})}$. The first row is identical to its shape in $\widehat{C}^{(\Theta_{\mathsf{cyclic}})}$. The return phase distributes elements of the two remaining rows in the reverse order, two elements at a time. The motivating intuition is that the slower return would further compensate for the imbalance in Θ_{cyclic}.

Greedy scheduling (Table 3(f)). The greedy scheduling algorithm, Θ_{greedy}, iteratively assigns a step to each group of chunks so as to balance the current success probabilities as much as possible. At each step, Θ_{greedy} constructs one new row of the execution chart $\widehat{C}^{(\mathsf{greedy})}$. Remember that after k steps, the probability that a chunk in group j will be interrupted is proportional to $\Pi_{i=1}^{k} \widehat{C}_{ij}^{(\mathsf{greedy})}$. The idea is to iteratively sort current column products and assign the smallest time-step to the largest product.

Numerical evaluation We ran our six scheduling heuristics on all problems where $g \in [2, 100]$, $n \in [2g, 1000]$, and g divides n: altogether 4032 instances. Table 4 presents two series of statistics. For heuristic $\overline{\Theta}$: The *Relative* series presents the ratio of $\mathsf{K}^{(\overline{\Theta})}$ on a given instance to the lowest $\mathsf{K}^{(\Theta)}$ value found for that instance among all the tested heuristics; the *Absolute* series presents the ratio of $\mathsf{K}^{(\overline{\Theta})}$ to $\mathsf{K}_{\min}$. Table 4 also records the *best-of* heuristic that, on each instance, runs all six heuristics and picks the best answer. Θ_{greedy} is clearly the best heuristic: it finds the best schedule for 83% of the instances, and its solutions are never more than 6% worse than the best solution found. More importantly, $\mathsf{K}^{(\Theta_{\mathsf{greedy}})}$ is never more than 23% larger than the lower bound $\mathsf{K}_{\min}$ and, on average, it is less than 7% larger than this bound. In fact, only $\Theta_{\mathsf{fat-snake}}$ happens sometimes to find better solutions than Θ_{greedy}; however, these improvements are marginal, as one can see by comparing the absolute performance of Θ_{greedy} and *best-of*. As a result of this experimentation, we retain only Θ_{greedy} as the exemplar of group schedules for the experiments of Section 6.

5.3 Choosing the Optimal Number of Chunks

We show now that one does not have to guess at the number n of chunks per computer. We begin to flesh out this remark by noting that we can easily obtain an explicit expression for the expected work completed by any group schedule Θ under the charged-initiation model, from Θ's analogous expectation under the free-initiation model.

Theorem 6 *Let $\mathcal{C}$ be a group schedule defined by the execution chart* $C_{i,j}\big|_{i\in\{1,\ldots,\mathsf{g}\},j\in\{1,\ldots,\mathsf{n}/\mathsf{g}\}}$. *If*

a) cyclic

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

$\mathsf{K}^{(\Theta_{\mathsf{cyclic}})} = 34104$

b) reverse

1	2	3	4	5
10	9	8	7	6
15	14	13	12	11
20	19	18	17	16

$\mathsf{K}^{(\Theta_{\mathsf{reverse}})} = 27284$

c) mirror

1	2	3	4	5
6	7	8	9	10
15	14	13	12	11
20	19	18	17	16

$\mathsf{K}^{(\Theta_{\mathsf{mirror}})} = 24396$

d) snake

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16

$\mathsf{K}^{(\Theta_{\mathsf{snake}})} = 25784$

e) fat − snake

1	2	3	4	5
14	12	10	8	6
15	13	11	9	7
16	17	18	19	20

$\mathsf{K}^{(\Theta_{\mathsf{fat-snake}})} = 24276$

f) greedy

1	2	3	4	5
10	9	8	7	6
15	14	13	12	11
20	19	18	16	17

$\mathsf{K}^{(\Theta_{\mathsf{greedy}})} = 24390$

Table 3. Illustrating the group schedules for $\mathsf{n} = 20$ and $\mathsf{g} = 4$. Here the most efficient schedule is $\Theta_{\mathsf{fat-snake}}$.

	Relative				Absolute				Success rate
	min	max	avg.	stdv.	min	max	avg.	stdv.	
Cyclic	1.1	3.786	2.143	0.664	1.1	3.786	2.239	0.592	00.00%
Reverse	1	1.295	1.055	0.065	1	1.295	1.117	0.061	12.42%
Mirror	1	2.468	1.504	0.393	1	2.468	1.575	0.338	12.37%
Snake	1	1.199	1.127	0.059	1	1.291	1.193	0.059	12.34%
Greedy	1	1.055	1.005	0.015	1	1.224	1.067	0.074	83.01%
Fat-snake	1	1.442	1.123	0.115	1	1.530	1.192	0.143	17.07%
Best-of	1	1	1	0	1	1.224	1.061	0.069	100.00%

Table 4. Statistics on the K value of all heuristics for $2 \leq g \leq 100$, $2g \leq n \leq 1000$, and g divides n (minimum, maximum, average value, and standard deviation over the 4032 instances).

$\mathsf{sl}^{(c)} \leq \min\{\mathsf{sl}, X - \mathsf{n}\varepsilon\}$, *then:*

$$E^{(c,\mathsf{n})}(\mathsf{sl}^{(c)}, \mathcal{C}) \;=\; \frac{\mathsf{sl}^{(c)}}{\mathsf{sl}^{(c)} + \mathsf{n}\varepsilon} E^{(f,\mathsf{n})}(\mathsf{sl}^{(c)} + \mathsf{n}\varepsilon, \mathcal{C}).$$

Now we determine the value of n, making only the assumption that the expectation of the group schedule within the charged-initiation model is a unimodal function of n. (It is quite natural to assume that this expectation is nondecreasing with n under the free-initiation model.) We can, therefore, use a binary search to seek the optimum value of n. Specifically, for each tested value m, we compare the values of the expectation for m and $m + 1$ to determine if the expectation is still increasing in m (so that m is smaller than the optimum n). The binary search can be safely performed in the interval $[1..X/\varepsilon]$.

6 Experiments

We have tried to assess the performance of our group heuristics by testing them on large simulated computing platforms that are subject to unrecoverable interruptions. Since all group heuristics were observed to have similar performance in our simulations [3], we only report here on the simulated performance of Θ_{greedy}. The source code and the complete set of results are available in [3].

We simulated randomly generated platforms comprising p computers. In all experiments, $\kappa = 1$, and the time at which each computer is interrupted falls randomly between 0 and 1, following a uniform distribution. The total workload has size $W_{tot} \in [1..p]$. $W_{tot} = 1$ represents the case when all computers can potentially complete all of the work before being interrupted; $W_{tot} = p$ represents the case when we can do no better than give one slice of size 1 to each computer, which then processes it until it is interrupted. The key parameters for the simulation are p, W_{tot}, the (common) chunk size cs, and the start-up cost ε. Several heuristics are compared:

H1-brute. This *brute replication heuristic* replicates the entire workload onto all computers. Each computer executes work in the order of receipt, starting from the first chunk, until it is interrupted.
H2-norep. This *no replication heuristic* distributes the work in a round-robin fashion, with no replication. Thus, each computer is allocated W_{tot}/p units of work (rounded by the chunk size).
H3-cyclicrep. This *cyclic replication heuristic* distributes the work in round-robin fashion, as does H2-norep, but it keeps distributing chunks, starting from chunk 1 again, until each computer has a total (local) workload of 1. Note that when n is a multiple of p, this heuristic is identical to H2-norep, since the chunks assigned to a computer after replication were already assigned to it previously.
H4-randomrep. This *random replication heuristic* distributes a local workload of 1 to each computer, but it chooses chunks randomly, to ensure that all chunks deployed on the same computer are distinct. However, the same chunk can be assigned to several computers.
H5-groupgreedy. This *group greedy heuristic* is the schedule Θ_{greedy} of Section 5.2.3. Since n may not be a multiple of g, the last group of computers may not have a full g chunks to process, in which case, we ignore this last group once its computers have been assigned as many time-steps as there are chunks in the group.
H6-omniscient. The *omniscient heuristic* is an idealized static heuristic that knows exactly when each computer is interrupted. This idealized knowledge obviates replication: each computer is statically allocated precisely as many chunks as it can process before its interruption, and only distinct chunks are sent. Of course no actual heuristic can beat this optimal omniscient heuristic.

In all experiments, we output the ratio W_{done}/W_{tot}, where W_{done} is the amount of work successfully completed by a heuristic. The experiments average the result obtained on 100 different random configurations of the system (interruption times). Only a significant subset of results is presented, the complete study being available in [3].

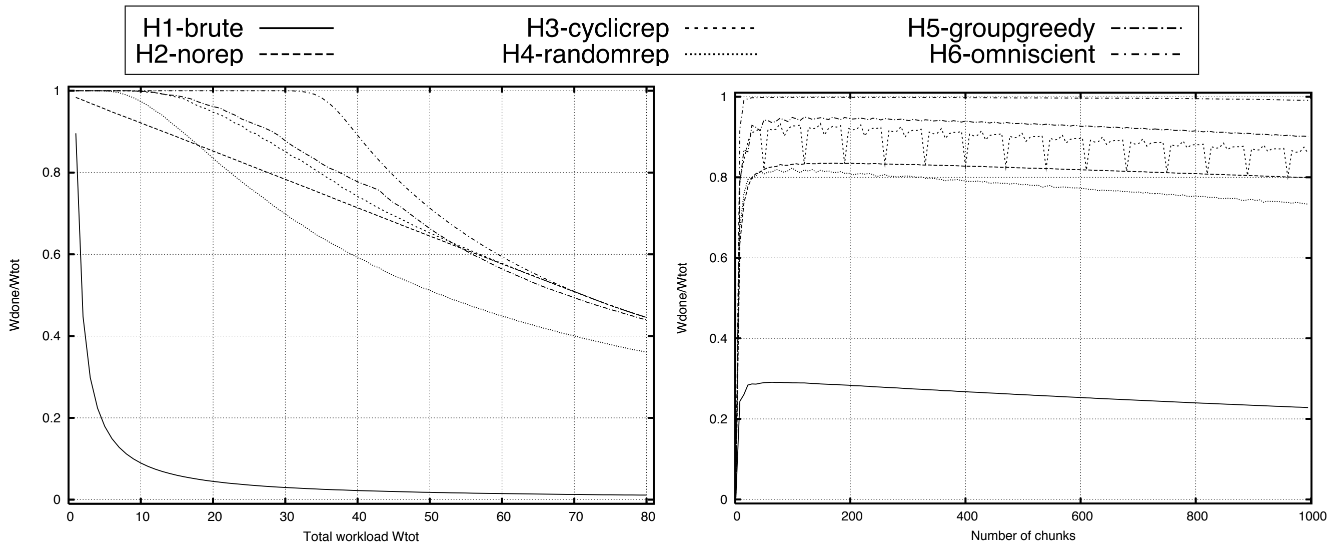
As expected, for all parameters, H6-omniscient always completes the most work, and H1-brute completes the least. The other heuristics are more comparable with each other. In all cases, H3-cyclicrep is better than H2-norep, but these two heuristics are equivalent when the total number of chunks is a multiple of p. H4-randomrep generally produces more work than H2-norep for small workloads, but it is not efficient for high loads since it may distribute the same chunk several times. Finally, the group greedy heuristic is the best one (aside from the optimal H6) most of the time, at least when the workload is not too large. When the workload increases, since the number of chunks is not necessarily a multiple of g, the handling of some chunks may not be optimized. Moreover, for a large workload, a cyclic distribution of the work will achieve a good result since not much redundant work can be done. This ranking is observed in Fig. 4(a), where the chunk size is $cs = 1/\mathsf{n} + \varepsilon$. The ranking is similar in all other tested sets of parameters [3].

These experiments also illustrate the impact of start-up cost on the perceived quality of a schedule. In particular, the simulations allow us to determine how many chunks should be used in order to maximize the expected work output, for a given setting; see, e.g., Fig. 4(b).

7 Conclusion

We have presented a model for studying the problem of scheduling large divisible workloads on p identical remote computers that are vulnerable (with the same risk function) to unrecoverable interruptions. Our goal has been schedules for allocating work to the computers and for scheduling the checkpointing of that work, in a manner that maximizes the expected amount of work completed by the remote computers. Most of our results assume that the risk of a computer being interrupted increases *linearly* with the amount of time that the computer has been available. We have completely solved this scheduling problem for $p = 1$ remote computer. Our solution provides exactly optimal schedules—whose expected work completion is exactly maximum—both for the free-initiation and charged-initiation models. For $p = 2$ remote computers, we provide schedules whose expected work output is asymptotically optimal, as the size of the workload grows without bound; we also provide guidelines for deriving exactly optimal schedules. The complexity of the development about 2 computers suggests that the general case of p remote computers will be prohibitively difficult, even with respect to deriving asymptotically optimal schedules. Therefore, we settle in this general case for deriving a number of well-structured heuristics, whose quality can be assessed via explicit expressions for their expected work outputs. Simulations suggest that our group-heuristics are clear winners in terms of performance. Extensive simulation experiments suggest that all our group-heuristics provide schedules with good expected work output in non trivial cases, that is, when there is work to replicate ($W < pX$) and the replication is not trivial ($W > X$).

Much remains to be done regarding this important problem, but three directions stand out as perhaps the major outstanding challenges. One of these is to extend our (asymptotic-) optimality results to a larger class of risk functions, thereby covering the range of situations that our work addresses.



(a) $p = 80, \mathsf{n} = 97, \varepsilon = 0.001$. (b) $p = 10, W_{tot} = 3, \varepsilon = 0.001$.

Figure 4. Performance of the different heuristics.

A second is to extend our study to include heterogeneous assemblages of remote computers, whose constituent computers differ in speed and other computational resources. When the assemblages are heterogeneous, but even when they are homogeneous, it would be significant to allow the assemblage's computers to be subject to differing probabilities of being interrupted.

Acknowledgment

The work of A. Benoit and Y. Robert was supported in part by the ANR StochaGrid project. The work of A. Rosenberg was supported in part by US NSF Grant CNS-0842578.

References

[1] 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. In *28th ACM Symp. on Theory of Computing*, pages 519–530, 1996.

[2] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. *IEEE Trans. Parallel Distributed Systems*, 16(3):207–218, 2005.

[3] A. Benoit, Y. Robert, A. L. Rosenberg, and F. Vivien. Static strategies for worksharing with unrecoverable interruptions. Research report RR2008-29, LIP, ENS Lyon, France, 2008.

[4] S. N. Bhatt, F. R. K. Chung, F. T. Leighton, and A. L. Rosenberg. An optimal strategies for cycle-stealing in networks of workstations. *IEEE Trans. Computers*, 46(5):545–557, 1997.

[5] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. of High Performance Computing Applications*, 15(3):200–222, 2001.

[6] M. Gallet, Y. Robert, and F. Vivien. Comments on "design and performance evaluation of load distribution strategies for multiple loads on heterogeneous linear daisy chain networks". *J. Parallel and Distributed Computing*, 68(7):1021–1031, 2008.

[7] S. D. L. in Parallel and D. Systems. *Veeravalli Bharadwaj and Debasish Ghose and Venkataraman Mani and Thomas G. Robertazzi*. Wiley-IEEE Computer Society Press, 1996.

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

[9] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. Seti@home-massively distributed computing for seti. *Computing in Science & Engineering*, 3(1):78–83, 2001.

[10] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *8th Intl. Conf. on Distr. Computing Systs. (ICDCS)*, pages 104–111, 1988.

[11] A. L. Rosenberg. Optimal schedules for cycle-stealing in a network of workstations with a bag-of-tasks workload. *IEEE Trans. Parallel Distrib. Syst.*, 13(2):179–191, 2002.

[12] S. White and D. Torney. Use of a workstation cluster for the physical mapping of chromosomes. *SIAM NEWS*, pages 14–17, Mar. 1993.

[13] J. Wingstrom and H. Casanova. Probabilistic allocation of tasks on desktop grids. In *Proceedings of the Workshop on Desktop Grids and Volunteer Computing Systems (PCGrid)*. IEEE CS Press, 2008.