Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

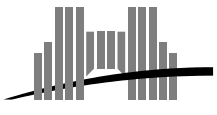
Static Strategies for Worksharing with Unrecoverable Interruptions

Anne Benoit ,
Yves Robert ,
Arnold L. Rosenberg ,
Frédéric Vivien

October 2008

Research Report N° 2008-29

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`



Static Strategies for Worksharing with Unrecoverable Interruptions

Anne Benoit , Yves Robert , Arnold L. Rosenberg , Frédéric Vivien

October 2008

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 are presented that suggest the intractability of such maximization for higher values of p, which motivates the development of heuristic approaches. Heuristics are developed that replicate work on several remote computers, in the hope of thereby decreasing the impact of work-killing interruptions.

Keywords: Fault-tolerance, scheduling, divisible loads, probabilities

Résumé

Une grande quantité de travail, qui peut être arbitrairement divisée, doit être traitée. Pour ce faire, nous avons p ordinateurs distants à notre disposition. Le problème est que ces ordinateurs sont susceptibles d'être vicimes d'interruptions, de probabilité connue, détruisant le travail en cours. On souhaite orchestrer le partage du travail entre les ordinateurs de manière à maximiser la quantité de travail que l'on peut espérer compléter. On étudie des stratégies visant à atteindre ce but en équilibrant l'envie d'effectuer souvent des sauvegardes, de manière à diminuer la quantité de travail risquant d'être perdue, et le désir d'éviter les changements de contexte nécessaires aux sauvegardes. On définit des stratégies qui maximise l'espérance de la quantité de travail faite quand il y a un seul ordinateur ($p = 1$). On montre des résultats qui suggèrent que la maximisation de cette espérance, pour des valeurs de p plus grandes, est intractable. On présente alors des heuristiques qui répliquent du travail dans l'espoir de minimiser l'impact des interruptions.

Mots-clés: Tolérance aux pannes, ordonancement, tâches divisibles, probabilités

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 the workload. The resulting cooperating computers may belong to a nearby or remote cluster (of "workstations"; cf. [30]), 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. [16, 21, 20]), global computing (cf. [18]), or volunteer computing (cf. [27]). In order to avoid unintended connotations concerning the organization of the remote computers, we avoid evocative terms such as "cluster" or "grid" 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 footsteps of sources such as [3, 10, 14, 23, 29, 34], which present 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., by allocating some work to more than one computer. Such a team-oriented viewpoint has appeared mainly in experimental studies (cf. [26]); ours is the first analytical study to adopt such a point of view.

The problem. We have a large computational workload whose constituent work is *divisible* in the sense that one can partition chuks of work into arbitrary granularities (cf. [13]). We also have access to $p \geq 1$ identical computers to help us compute the workload via *worksharing* (wherein the owner of the workload allocates work to remote computers that are idle; cf. [35]).

> 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, whose computers may differ in power and speed.

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. [3, 14, 31, 32, 34]). The scheduling tool that we employ to cope with these interruptions is *work replication*, the allocation of chunks 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.[1]

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.

[1] As in [14, 31, 34], our 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.

Therefore, we implicitly assume that we are dealing with applications for which even partial output is meaningful, e.g., Monte-Carlo simulations.

Three challenges. The challenges of scheduling our workload on interruptible remote computers can be described in terms of three dilemmas. The first two apply even to each remote computer individually.[2]

1. **If** we send each remote computer a large amount of work with each transmission,
   then we both decrease the overhead of packaging work-containing messages and maximize the opportunities for "parallelism" within the assemblage of remote computers,
   but we thereby maximize our vulnerability to losing work because of a remote computer's being interrupted.

On the other hand,

2. **If** we send each remote computer a small amount of work with each transmission,
   then we minimize our vulnerability to interruption-induced losses,
   but we thereby maximize message overhead and minimize the opportunities for "parallelism" within the assemblage of remote computers.

The third dilemma arises only when there are at least two remote computers.[3]

3. **If** we *replicate work,* by sending the same work to more than one remote computer,
   then we lessen our vulnerability to interruption-induced losses,
   but we thereby minimize both the opportunities for "parallelism" and the expected productivity advantage from having access to the remote computers.

Approaches to the challenges. *(1) "Chunking" our workload.* We cope with the first two dilemmas by sending work allocations to the remote computers as a sequence of *chunks*[4] rather than as a single block to each computer. This approach, which is advocated in [14, 31, 32, 34], allows each computer to *checkpoint* at various times and, thereby, to protect some of its work from the threat of interruption. *(2) Replicating work.* We allocate certain chunks that are especially vulnerable to being interrupted to more than one remote computer in order to enhance their chances of being computed successfully. We use work replication judiciously, in deference to the third dilemma.

> Under our model, the risk of a computer's being interrupted increases as the computer operates, whether it works on our computation or not. This assumption models, e.g., interruptions from hardware failures or from returning owners in cycle-stealing scenarios. Thus, certain chunks of our workload are more vulnerable to being interrupted than others. To wit, the first "round" of allocated chunks involves our first use of the remote computers; hence, these chunks are less likely to be interrupted than are the chunks that are allocated in the second "round": the remote computers will have been operating longer by the time the second "round" occurs. In this manner, the second-"round" chunks are less vulnerable than the third-"round" chunks, and so on.

[2] We put "parallelism" in quotes when stating these dilemmas because remote computers are (usually) not synchronized, so they do not truly operate *in parallel*.

[3] The pros and cons of work replication are discussed in [26].

[4] We use the generic "chunk" instead of "task" to emphasize tasks' divisibility: by definition, divisible workloads do not have atomic tasks.

Because communication to remote computers is likely to be costly in time and overhead, we limit such communication by orchestrating work replication 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 thereby prevent *our* computer, which is the server in the studied scenario, from becoming a communication bottleneck when there are many interruptions. Our cost concerns are certainly justified when we access remote computers over the Internet, but also when accessing computers over a variety of common local-area networks (LANs). Moreover, as noted earlier, we get a good "return" from our conservative work replication, by increasing the expected amount of work done by the remote computers.

In summation, we assume: that we know the instantaneous probability that a remote computer will have been interrupted by time t; that this probability is the same for all remote computers; that the probability increases linearly with the amount of time that the computer has been available. These assumptions, which we share with [14, 31, 34], seem to be necessary in order to derive scheduling strategies that are *provably* optimal. As suggested in these sources (cf. footnote 1), one can use approximate knowledge of these probabilities, obtained, say, via trace data, but this will clearly weaken our performance claims for our schedules. Also as noted earlier, the challenge of allowing individual computers to have different probabilities must await a sequel to the current study.

Related work. The literature contains relatively few rigorously analyzed scheduling algorithms for interruptible "parallel" computing in assemblages of computers. Among those we know of, only [3, 14, 31, 32, 34] deal with an *adversarial* model of interruptible computing. One finds in [3] a randomized scheduling strategy which, with high probability, completes within a logarithmic factor of the optimal fraction of the initial workload. In [14, 31, 32, 34], the scheduling problem is viewed as a game against a malicious adversary who seeks to interrupt each remote computer in order to kill all work in progress and thereby minimize the work amount of completed during an interruptible computation. Among the experimental sources, [37] 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 very large literature on scheduling divisible workloads on assemblages of computers that are not vulnerable to interruption. We refer the reader first to [13] and its myriad intellectual progeny; not all of these sources share the current study's level of detailed argumentation. One finds in [2], its precursor [33], and its accompanying experimental study [1], an intriguing illustration of the dramatic impact on the scheduling problem for heterogeneous assemblages of having to account for the transmission of the output generated by the computation; a different aspect of the same observation is noted in [9]. Significant preliminary results about assemblages in which communication links, as well as constituent computers, are heterogeneous appear in [9]. Several studies focus on scheduling divisible computations but focus on algorithmically simpler computations whose tasks produce no output. A near-optimal algorithm for such scheduling appears in [38] under a simplified model, in which equal-size chunks of work are sent to remote computers at a frequency determined by the computers' powers. The body of work exemplified by [11, 12, 13, 17, 19] and sources cited therein allow heterogeneity in both communication links and computers, but schedule outputless tasks, under a simple communication model. (It is worth noting that one consequence of a linear, rather

than affine communication cost model is that it can be advantageous to distribute work in *many* small pieces, rather than in a few large chunks; cf. [12, 38].) A significant study that shares our focus on tasks having equal sizes and complexities, but that allows workstations to redistribute allocated tasks, appears in [5, 7]. Under the assumption of unit computation time per task, these sources craft linear-programming algorithms that optimize the steady-state processing of tasks. The distribution of inputs and subsequent collection of results form an integral part of [2, 9]; these problems are studied as independent topics in [6].

Even the subset of the divisible workload literature that focuses on collective communication in assemblages of computers is enormous. Algorithms for various collective communication operations appear in [4, 25]. One finds in [22] approximation algorithms for a variant of broadcasting under which receipt of the message "triggers" a "personal" computation whose cost is accounted for within the algorithm.

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 [28] as an exemplar of the former type of study and to [36] as an exemplar of the latter.

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 begin with W units of divisible work that we wish to execute on an assemblage of $p \geq 1$ identical computers that are susceptible to unrecoverable interruptions that "kill" all work currently in progress. All computers share the same instantaneous probability of being interrupted, and this probability increases with the amount of time the computer has been operating (whether working on our computation or not). We know this probability exactly.[5]

> Because we deal with a single computational application and identical computers, we lose no generality by expressing our results in terms of units of work, rather than the execution time of these units. We paraphrase the following explanation from [2], which uses a similar convention.
>
> Our results rely only on the fact that all work units have the same size and complexity: formally, there is a constant $c > 0$ such that executing w units of work takes cw time units. *The work units' (common) complexity can be an arbitrary function of their (common) size:* c is simply the ratio of the fixed size of a work unit to the complexity of executing that amount of work.

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

[5] As stated earlier, our analyses can be modified to accommodate probabilities that are known only statistically.

one chunk to each computer in the assemblage. Instead, we "protect" our workload as best we can, by:

- partitioning it into chunks of identical (not-necessarily integer) size—which will be the unit of work that we allocate to the computers
- prescribing a schedule for allocating chunks to computers
- allocating some chunks to more than one computer, as a divisible-load mode of work replication.

As noted in the Introduction, we treat intercomputer communication as a resource to be used very conservatively—which is certainly justified when communication is over the Internet, and often when communication is over common local-area networks (LANs). Specifically, we 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. We find that when p is small, chopping work into *equal-size* chunks is either optimal (when $p = 1$) or asymptotically optimal (when $p = 2$). Motivated by this discovery, we simplify the heuristic schedules we derive for general values of p, by restricting attention to schedules that employ equal-size chunks. We select the common size of the chunks and the regimen for allocating them to the computers in the assemblage, based on the value of p and of the (common) probability (or, rate) of interruption.

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."

> *Recall that 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 in our homogeneous setting; it is personalized to each computer in a heterogeneous setting.

Of course, $Pr(w)$ increases with w; we assume that we know its value exactly: see, however, footnote 1.

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 conditional probability that a remote computer will be interrupted during the next w "time units," given that it has not been interrupted during the first s "time units." Thus, $Pr(w) = Pr(0, w)$ and $Pr(s, w) = Pr(s + w) - Pr(s)$.

We let[6] $\kappa \in (0, 1]$ be a constant that weights our probabilities in a way that reflects the common size of our work chunks. We illustrate the role of κ as we introduce two specific common risk functions Pr, the first of which is our focus in the current study.

[6] As usual, $(a, b]$ (resp., $[a, b]$) denotes the real interval $\{x \mid a < x \leq b\}$ (resp., $\{x \mid a \leq x \leq b\}$).

Linearly increasing risk. The risk function that will be the main focus of our study is $Pr(w) = \kappa w$. It is the most natural model in the absence of further information: the failure risk grows linearly, in proportion to the time spent, or equivalently to the amount of work done. This linear model covers a wide range of cycle-stealing scenarios, but also situations when interruptions are due to hardware failures.

In this case, we have the density function

$$dPr \;=\; \begin{cases} \kappa dt & \text{for } \; t \in [0, 1/\kappa] \\ 0 & \text{otherwise} \end{cases}$$

so that

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

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

Geometrically decaying lifespan. A commonly studied risk function, which models a variety of common "failure" scenarios, is $Pr(w) = 1 - \exp^{-\kappa w}$, wherein the probability of a computer's surviving for one more "time step" decays geometrically. More precisely,

$$Pr(w, 1) = Pr(w + 1) - Pr(w) \;=\; (1 - \exp^{-\kappa(w+1)}) - (1 - \exp^{-\kappa w}) \;=\; (1 - \exp^{-\kappa}) \exp^{-\kappa w} .$$

One might expect such a risk function, for instance, when interruptions are due to someone's leaving work for the day; the longer s/he is absent, the more likely it is that s/he is gone for the night.

In this case, we have the density function $dPr = \kappa \exp^{-\kappa t} dt$, so that

$$Pr(s, w) \;=\; \int_s^{s+w} \kappa \exp^{-\kappa t} dt \;=\; \exp^{-\kappa s}(1 - \exp^{-\kappa w}).$$

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. To this end, we focus on a workload consisting of W work units, and we let jobdone be the random variable whose value is the number of work units that the assemblage executes successfully under a given scheduling regimen. Stated 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 they account for the way a chunk defines the notion of a "time unit." The actual time for processing a chunk of work has several components:

- There is the overhead for transmitting the chunk to the remote computer. This may be a trivial amount of actual time if one must merely set up the communication, or it may be a quite significant amount if one must, say, encode the chunk before transmitting it. In the latter case, the overhead can be proportional to the chunk size.
- There is the time to actually transmitting the chunk, which is proportional to the chunk size.
- There is the actual time that the remote computer spends executing the chunk, which, by assumption, is proportional to the chunk size.
- There is the time that the remote computer spends checkpointing after computing a chunk. This may be a trivial amount of actual time—essentially just a context switch—if the chunk creates little output (perhaps just a YES/NO decision), or it may be a quite significant amount if the chunk creates a sizable output (e.g., a matrix inversion).

In short, there are two classes of time-costs, those that are proportional to the size of a chunk and those that are fixed constants. It simplifies our formal analyses to fold the first class of time-costs into a single quantity that is proportional to the size of a chunk and to combine the second class into a single fixed constant. 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 second cost! One could, in fact, strive for adaptive schedules that change their strategies depending on the changing ratios between chunk sizes and fixed costs. However, for the reasons discussed earlier, we prefer to seek *static* scheduling strategies, at least until we have a well-understood arsenal of tools for scheduling interruptible divisible workloads. Therefore, we perform the current study with two fixed cost models, striving for optimal schedules under each. (1) The *free-initiation model* is characterized by *not* charging the owner of the workload a per-chunk cost. This model focuses on situations wherein the fixed costs are negligible compared to the chunk-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, which assesses no per-chunk cost, is much the easier of our two models to analyze. The results obtained using this model approximate reality well when one knows *a priori* that chunks must be large. One situation that mandates large chunks is when communication is over the Internet, so that one must have every remote computer do a substantial amount of the work in order to amortize the time-cost of message transmission (cf. [27]). In such a situation, one will keep 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 predictably good *bounds* on the expected value of $\mathsf{jobdone}$ under the charged-initiation model, in situations where such bounds are prohibitively hard to derive directly; cf. Theorem 1.

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

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

Let us illustrate this model via three simple calculations of $E^{(\mathrm{f})}(\mathsf{jobdone}, \Theta)$. In these calculations, the regimen Θ allocates the whole workload and deploys it on a single computer. To enhance legibility, let the phrase "under Θ" within "$Pr(\mathsf{jobdone} \geq u$ under $\Theta)$" be specified implicitly by context.

Deploying the workload as a single chunk. Under regimen Θ_1 the whole workload is deployed as a single chunk on a single computer. By definition, $E^{(\mathrm{f})}(W, \Theta_1)$ for an arbitrary risk function Pr is given by

$$E^{(\mathrm{f})}(W, \Theta_1) \;=\; W\,(1 - Pr(W))\,. \tag{2}$$

Deploying the workload in two chunks. Regimen Θ_2 specifies how the workload is split into the two chunks of respective sizes $\omega_1 > 0$ and $\omega_2 > 0$, where $\omega_1 + \omega_2 = W$. The following derivation determines $E^{(\mathrm{f})}(W, \Theta_2)$ for an arbitrary risk function Pr.

$$\begin{aligned} E^{(\mathrm{f})}(W, \Theta_2) &= \int_0^{\omega_1} Pr(\mathsf{jobdone} \geq u)du + \int_{\omega_1}^{\omega_1+\omega_2} Pr(\mathsf{jobdone} \geq u)du \\ &= \int_0^{\omega_1} Pr(\mathsf{jobdone} \geq \omega_1)du + \int_{\omega_1}^{\omega_1+\omega_2} Pr(\mathsf{jobdone} \geq \omega_1 + \omega_2)du \\ &= \omega_1(1 - Pr(\omega_1)) \;+\; \omega_2(1 - Pr(\omega_1 + \omega_2)). \end{aligned} \tag{3}$$

Deploying the workload in n chunks. Continuing the reasoning of the cases $n = 1$ and $n = 2$, we finally obtain the following general expression for $E^{(\mathrm{f})}(W, \Theta_n)$ for an arbitrary risk function Pr, when Θ_n partitions the whole workload into n chunks of respective sizes $\omega_1 > 0$, $\omega_2 > 0$, $\ldots$, $\omega_n > 0$ such that $\omega_1 + \cdots + \omega_n = W$.

$$\begin{aligned} E^{(\mathrm{f})}(W, \Theta_n) &= \int_0^{\omega_1} Pr(\mathsf{jobdone} \geq u)du + \int_{\omega_1}^{\omega_1+\omega_2} Pr(\mathsf{jobdone} \geq u)du \\ &\quad + \cdots + \int_{\omega_1+\cdots+\omega_{n-1}}^{\omega_1+\cdots+\omega_{n-1}+\omega_n} Pr(\mathsf{jobdone} \geq u)du \\ &= \int_0^{\omega_1} Pr(\mathsf{jobdone} \geq \omega_1)du + \int_{\omega_1}^{\omega_1+\omega_2} Pr(\mathsf{jobdone} \geq \omega_1 + \omega_2)du \\ &\quad + \cdots + \int_{\omega_1+\cdots+\omega_{n-1}}^{\omega_1+\cdots+\omega_{n-1}+\omega_n} Pr(\mathsf{jobdone} \geq \omega_1 + \cdots + \omega_n)du \\ &= \omega_1(1 - Pr(\omega_1)) \;+\; \omega_2(1 - Pr(\omega_1 + \omega_2)) \\ &\quad + \cdots + \;\omega_n(1 - Pr(\omega_1 + \cdots + \omega_n)). \end{aligned} \tag{4}$$

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_n that maximizes $E^{(\mathrm{f})}(W, \Theta_n)$. However, we have a more ambitious goal, which is motivated by the following observation.

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 this expectation by making the last deployed chunk even a tiny bit smaller than needed to deploy all W units of work.

We shall see the preceding observation in operation 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 (rather than to exactly W as in the preceding paragraphs),
- 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 $E^{(\mathrm{f})}(W, n)$:

$$E^{(\mathrm{f})}(W, n) \;=\; \max\{\omega_1(1 - Pr(\omega_1)) + \cdots + \omega_n(1 - Pr(\omega_1 + \cdots + \omega_n))\},$$

where the maximization is over all n-tuples of positive chunk sizes that sum to at most W:

$$\{\omega_1 > 0,\ \omega_2 > 0, \ldots,\ \omega_n > 0\} \quad \text{such that} \quad \omega_1 + \omega_2 + \cdots + \omega_n \le W$$

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, *the charged-initiation model often allows one to determine analytically the best numbers of chunks and of "rounds"* (when there are multiple remote computers). Under the charged-initiation model, an explicit fixed cost, ε units of work, is added to the cost of computing of each chunk, to reflect the overhead when a remote computer begins processing each new chunk. This overhead could reflect the cost of the communication needed to transmit the chunk to the remote computer, or it could reflect the cost of the interrupt plus context switch incurred as the remote computer checkpoints some work. Under this model, the expected value of $\mathsf{jobdone}$ under a given scheduling regimen Θ, denoted $E^{(\mathrm{c})}(\mathsf{jobdone}, \Theta)$, the superscript "c" denoting "charged(-initiation)" is

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

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. The free-initiation model enables us to obtain bounds on the charged-initiation model, in the manner indicated by the following theorem.

Theorem 1. (Charged-initiation output vs. Free-initiation output)
The optimal n-chunk expected value of jobdone *under the* charged-*initiation model, $E^{(\mathrm{c})}(W,n)$, is never greater than the analogous quantity, $E^{(\mathrm{f})}(W,n)$, for the* free-*initiation model; the latter quantity cannot exceed the former by more than n times the size of the overhead ε. Symbolically,*

$$E^{(\mathrm{f})}(W,n) \;\geq\; E^{(\mathrm{c})}(W,n) \;\geq\; E^{(\mathrm{f})}(W,n) - n\varepsilon. \tag{5}$$

Proof. The lefthand bound in (5) is obvious, because risk functions are nondecreasing—so that, for any given scheduling regimen, the expected value of jobdone under the charged-initiation model cannot exceed the expected value under the free-initiation model.

To derive the righthand bound in (5), let us focus on the optimal scheduling regimen Θ under the free-initiation model. Θ schedules the load via n chunks of size $\omega_1 > 0, \ldots, \omega_n > 0$. Let $\omega_i' = \max\{0, \omega_i - \varepsilon\}$, and let Θ' denote the scheduling regimen that executes n chunks of sizes ω_1', ..., ω_n', in that order (except that zero-length chunks are not really executed). We account for these zero-length chunks in the following equations, via the function

$$\mathbb{1}_{\omega_i'} \;=\; \begin{cases} 1 & \text{if } \omega_i' \neq 0 \\ 0 & \text{if } \omega_i' = 0. \end{cases}$$

Since Θ' implicitly specifies a scheduling regimen for the charged-initiation model when using $\leq n$ chunks, the expected value of jobdone under Θ' obviously cannot exceed the expected value under the *best* scheduling regimen for the charged-initiation model when using $\leq n$ chunks. Therefore,

$$\begin{aligned}
E^{(\mathrm{c})}(W,n) &\geq E^{(\mathrm{c})}(W,\Theta') \\
&= \sum_{i=1}^{n} \omega_i' \left(1 - Pr\left(\sum_{i=1}^{n} \omega_i' + \varepsilon \mathbb{1}_{\omega_i'}\right)\right) \\
&= \sum_{i=1}^{n} \omega_i' \left(1 - Pr\left(\sum_{i=1}^{n} \mathbb{1}_{\omega_i'}(\omega_i' + \varepsilon)\right)\right) \\
&= \sum_{i=1}^{n} \omega_i' \left(1 - Pr\left(\sum_{i=1}^{n} \mathbb{1}_{\omega_i'}\omega_i\right)\right) \\
&\geq \sum_{i=1}^{n} \omega_i' \left(1 - Pr\left(\sum_{i=1}^{n} \omega_i\right)\right) \\
&= \sum_{i=1}^{n} \omega_i \left(1 - Pr\left(\sum_{i=1}^{n} \omega_i\right)\right) - \varepsilon \sum_{i=1}^{n} \left(1 - Pr\left(\sum_{i=1}^{n} \omega_i\right)\right) \\
&\geq E^{(\mathrm{f})}(W,\Theta) - n\varepsilon \\
&= E^{(\mathrm{f})}(W,n) - n\varepsilon
\end{aligned}$$

which yields the righthand bound. □

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 that is subject to the linear risk of interruption: $Pr(w) = \min(1, \kappa w)$.

Some of the results we derive bear a striking similarity to their analogues in [14], despite certain substantive differences in models.

3.1 An Optimal Schedule under the Free-Initiation Model

We begin with a simple illustration of why the risk of losing work because of an interruption must affect our scheduling strategy, even when there is only one remote computer and even when dispatching a new chunk of work incurs no cost, *i.e.,* under the free-initiation model.

When the amount of work W is no larger than X (recall that $\kappa = 1/X$ is the constant that accounts for the size of the work-unit), then instantiating the linear risk function, $Pr(W) = \kappa W$, in (2) shows that the expected amount of work achieved when deploying the entire workload in a single chunk is

$$E^{(\mathrm{f})}(W, \Theta_1) \;=\; W - \kappa W^2.$$

Similarly, instantiating this risk function in (3) shows that the expected amount of work achieved when deploying the entire workload using two chunks, of respective sizes $\omega_1 > 0$ and $\omega_2 > 0$ is (recalling that $\omega_1 + \omega_2 = W$)

$$\begin{aligned} E^{(\mathrm{f})}(W, \Theta_2) &= \omega_1(1 - \omega_1\kappa) + \omega_2(1 - (\omega_1 + \omega_2)\kappa)) \\ &= W - (\omega_1^2 + \omega_1\omega_2 + \omega_2^2)\kappa \\ &= W - W^2\kappa + \omega_1\omega_2\kappa. \end{aligned}$$

We observe that

$$E^{(\mathrm{f})}(W, \Theta_2) - E^{(\mathrm{f})}(W, \Theta_1) \;=\; \omega_1\omega_2\kappa \;>\; 0.$$

Thus, as one would expect intuitively: *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.*

Continuing with the preceding reasoning, we can actually characterize the optimal—i.e., expectation-maximizing—schedule for any fixed number of chunks. (We thereby also identify 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. (One remote computer: free-initiation model)
Say that one wishes to deploy $W \in [0, X]$ units of work to a single remote computer in at most n chunks, for some positive integer n. In order to maximize the expectation of jobdone*, one should have all n chunks share the same size, namely, 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.

Note that for fixed W, $E^{(\mathrm{f})}(W,n)$ increases with n.

Proof. Let us partition the W-unit workload into $n+1$ chunks, of respective sizes $\omega_1 \geq 0$, $\ldots$, $\omega_n \geq 0$, $\omega_{n+1} \geq 0$, with the intention of deploying the first n of these chunks.

> (a) Our assigning the first n chunks *nonnegative,* rather than *positive* sizes affords us a convenient way to talk about "at most n chunks" using only the single parameter n. (b) By creating $n+1$ chunks rather than n, we allow ourselves to hold back some work in order to avoid what would be a certain interruption of the nth chunk. Formally, exercising this option means making ω_{n+1} positive; declining the option—thereby deploying all W units of work—means setting $\omega_{n+1} = 0$.

Each specific such partition specifies an n-chunk schedule Θ_n. Our challenge is to choose the sizes of the $n+1$ chunks in a way that maximizes $E^{(\mathrm{f})}(W,\Theta_n)$. To simplify notation, let $Z = \omega_1 + \cdots + \omega_n$ denote the portion of the entire workload that we actually deploy.

Extending the reasoning from the cases $n=1$ and $n=2$, one obtains easily from (4) the expression

$$\begin{aligned} E^{(\mathrm{f})}(W,\Theta_n) &= \omega_1(1-\omega_1\kappa) + \omega_2(1-(\omega_1+\omega_2)\kappa) + \cdots + \omega_n(1-(\omega_1+\cdots+\omega_n)\kappa) \\ &= Z - Z^2\kappa + \left[\sum_{1\leq i<j\leq n} \omega_i\omega_j\right]\kappa. \end{aligned} \tag{6}$$

Standard arguments show that the bracketed sum in (6) is maximized when all ω_i's share the common value Z/n, in which case, the sum achieves the value $\frac{1}{n^2}\binom{n}{2}Z^2\kappa$. Since maximizing the sum also maximizes $E^{(\mathrm{f})}(W,\Theta_n)$, simple arithmetic yields:

$$E^{(\mathrm{f})}(W,\Theta_n) = Z - \frac{n+1}{2n}Z^2\kappa.$$

Viewing this expression for $E^{(\mathrm{f})}(W,\Theta_n)$ as a function of Z, we note that the function is unimodal, increasing until $Z = \dfrac{n}{(n+1)\kappa}$ and decreasing thereafter. Setting this value for Z, gives us the maximum value for $E^{(\mathrm{f})}(W,\Theta_n)$, i.e., the value of $E^{(\mathrm{f})}(W,n)$. The theorem follows. □

3.2 An Optimal Schedule under the Charged-Initiation Model

Under the *charged-initiation* model—i.e., on a computing platform wherein processing a new chunk of work (for transmission or checkpointing) *does* incur a cost (that we must account for)—deriving the optimal strategy becomes dramatically more difficult, even when there is only one remote computer and even when we know *a priori* how many chunks we wish to employ.

Theorem 3. ***(One remote computer: charged-initiation model)***
Say that one wishes to deploy $W \in [0,X]$ units of work, where $X \geq \varepsilon$, to a single remote

computer in at most n chunks, for some positive integer n. 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* $E^{(\mathrm{c})}(\mathsf{jobdone})$ *specifies* $m = \min\{n, n_1, n_2\}$ *chunks: the first has size*[7]

$$\omega_{1,m} \;=\; \frac{Z}{m} + \frac{m-1}{2}\varepsilon$$

where

$$Z \;=\; \min\left\{W,\; \frac{m}{m+1}X - \frac{m}{2}\varepsilon\right\}; \tag{7}$$

and the $(i+1)$th chunk inductively has size

$$\omega_{i+1,m} \;=\; \omega_{i,m} \;-\; \varepsilon.$$

In expectation, this schedule completes

$$E^{(\mathrm{c})}(W,n) \;=\; Z - \frac{m+1}{2m}Z^2\kappa - \frac{m+1}{2}Z\varepsilon\kappa + \frac{(m-1)m(m+1)}{24}\varepsilon^2\kappa \tag{8}$$

units of work.

Proof. We proceed by induction on the number n of chunks we want to partition our W units of work into. We denote by $\mathcal{E}^{(\mathrm{c})}_{\mathrm{opt}}(W,n)$ the maximum expected amount of work that a schedule can complete under such a partition.

Focus first on the case $n = 1$. When work is allocated in a single chunk, the maximum expected amount of total work completed is, by definition:

$$\mathcal{E}^{(\mathrm{c})}_{\mathrm{opt}}(W,1) \;=\; \max_{0\le\omega_{1,1}\le W} \mathcal{E}^{(\mathrm{c})}(\omega_{1,1}) \quad \text{where} \quad \mathcal{E}^{(\mathrm{c})}(\omega_{1,1}) \;=\; \max_{0\le\omega_{1,1}\le W} \omega_{1,1}(1-(\omega_{1,1}+\varepsilon)\kappa).$$

We determine the optimal size of $\omega_{1,1}$ by viewing this quantity as a variable in the closed interval $[0, W]$ and maximizing $\mathcal{E}^{(\mathrm{c})}(\omega_{1,1})$ symbolically. We thereby find that $\mathcal{E}^{(\mathrm{c})}(\omega_{1,1})$ is maximized by setting

$$\omega_{1,1} \;=\; \min\left\{W, \frac{1}{2\kappa} - \frac{\varepsilon}{2}\right\},$$

so that

$$\mathcal{E}^{(\mathrm{c})}_{\mathrm{opt}}(W,1) \;=\; \begin{cases} \dfrac{1}{4\kappa} - \dfrac{\varepsilon}{2} + \dfrac{\varepsilon^2}{4}\kappa & \text{if } W > \dfrac{1}{2\kappa} - \dfrac{\varepsilon}{2}, \\ W - W^2\kappa - W\varepsilon\kappa & \text{otherwise.} \end{cases}$$

(Note that $\omega_{1,1}$ has a non-negative size because of the natural hypothesis that $X \ge \varepsilon$.)

We now proceed to general values of n by induction. We begin by assuming that the conclusions of the theorem have been established for the case when the workload is split into $n \ge 1$ positive-size chunks. We also assume that n is no greater than $\min\{n_1, n_2\}$. In other words, we assume that any optimal solution with at most n chunks used n positive-size chunks.

As our first step in analyzing how best to deploy $n+1$ positive-size chunks, we note that the only influence the first n chunks of work have on the probability that the last chunk will be computed successfully is in terms of their cumulative size.

[7] The second subscript of ω reminds us how many chunks the workload is divided into.

Let us clarify this last point, which follows from the failure probability model. Denote by A_n the cumulative size of the first n chunks of work in the expectation-maximizing $(n+1)$-chunk scenario; i.e., $A_n = \sum_{i=1}^{n} \omega_{i,n+1}$. Once A_n is specified, the probability that the remote computer will be interrupted while working on the $(n+1)$th chunk depends only on the value of A_n, *not* on the way the A_n units of work have been divided into chunks.

This fact means that once one has specified the cumulative size of the workload that comprises the first n chunks, the best way to partition this workload into chunks is as though it were the only work in the system, i.e., as if there were no $(n+1)$th chunk to be allocated. Thus, one can express $\mathcal{E}_{\text{opt}}(W, n+1)$ in terms of A_n (whose value must, of course, be determined) and $\mathcal{E}_{\text{opt}}(A_n, n)$, via the following maximization.

$$\mathcal{E}_{\text{opt}}(W, n+1) \;=\; \max\Big\{\mathcal{E}_{\text{opt}}(A_n, n) \;+\; \omega_{n+1,n+1}\,(1 - (A_n + \omega_{n+1,n+1} + (n+1)\varepsilon)\,\kappa)\Big\},$$

where the maximization is over all values for A_n in which

$$\begin{array}{rcll} A_n & > & 0 & \text{allowing for the } n \text{ previous chunks} \\ \omega_{n+1,n+1} & \geq & 0 & \text{allowing for an } (n+1)\text{th chunk} \\ A_n + \omega_{n+1,n+1} & \leq & W & \text{because the total workload has size } W \\ A_n + \omega_{n+1,n+1} + (n+1)\varepsilon & \leq & X & \text{reflecting the risk and cost models} \end{array}$$

The last of these inequalities acknowledges that the remote computer is certain to be interrupted (with probability 1) before it can complete the $(n+1)$th chunk of work, if its overall workload is no smaller than $X - (n+1)\varepsilon$.

We now have two cases to consider, depending on the size of A_n.

Case 1: $A_n < \dfrac{n}{n+1}X - \dfrac{n}{2}\varepsilon$.

By assumption, the expectation-maximizing regimen deploys A_n units of work via its first n chunks. By induction, expression (8) tells us that the expected amount of work completed by deploying these A_n units is

$$\mathcal{E}^{(c)}_{\text{opt}}(A_n, n) \;=\; A_n - \frac{n+1}{2n}A_n^2\kappa - \frac{n+1}{2}A_n\varepsilon\kappa + \frac{(n-1)n(n+1)}{24}\varepsilon^2\kappa.$$

Let Z denote the total work that is actually allocated: $Z = A_n + \omega_{n+1,n+1}$. In the following calculations, we write $\omega_{n+1,n+1}$ as $Z - A_n$, in order to represent the $(n+1)$-chunk scenario entirely via quantities that arise in the n-chunk scenario.

We focus on

$$\begin{aligned} \mathcal{E}^{(1)}(A_n, \omega_{n+1,n+1}) &= \mathcal{E}_{\text{opt}}(A_n, n) + (Z - A_n)\,(1 - (Z + (n+1)\varepsilon)\,\kappa) \\ &= \left(A_n - \frac{n+1}{2n}A_n^2\kappa - \frac{n+1}{2}A_n\varepsilon\kappa + \frac{(n-1)n(n+1)}{24}\varepsilon^2\kappa\right) \\ &\quad + (Z - A_n)\,(1 - (Z + (n+1)\varepsilon)\,\kappa) \\ &= \left(Z + \frac{n+1}{2}\varepsilon\right)A_n\kappa - \frac{n+1}{2n}A_n^2\kappa \\ &\quad + Z(1 - (Z + (n+1)\varepsilon)\kappa) + \frac{(n-1)n(n+1)}{24}\varepsilon^2\kappa. \end{aligned}$$

For a given value of Z, we look for the best value for A_n using the preceding expression. We note first that

$$\frac{\partial \mathcal{E}^{(1)}(A_n, \omega_{n+1,n+1})}{\partial A_n} \;=\; -\frac{n+1}{n} A_n \kappa + Z\kappa + \frac{n+1}{2}\varepsilon\kappa.$$

We note next that, for fixed Z, the quantity $\mathcal{E}^{(1)}(A_n, \omega_{n+1,n+1})$ begins to increase with A_n and then decreases. The value for A_n that maximizes this expectation, which we denote $A_n^{(\text{opt})}$, is

$$A_n^{(\text{opt})} \;=\; \min\left\{W, \frac{n}{n+1}Z + \frac{n}{2}\varepsilon\right\}.$$

When $W \leq (n/(n+1))Z + \frac{1}{2}n\varepsilon$, $A_n^{(\text{opt})} = W$, meaning that the $(n+1)$th chunk is empty, and the schedule does *not* optimize the expected work. (In the charged-initiation model an empty chunk decreases the overall probability). Consequently, we focus *for the moment* on the case

$$A_n^{(\text{opt})} \;=\; \frac{n}{n+1}Z + \frac{n}{2}\varepsilon \tag{9}$$

(thereby assuming that $W \geq (n/(n+1))Z + \frac{1}{2}n\varepsilon$). Therefore, we have

$$\mathcal{E}^{(1)}(A_n^{(\text{opt})}, \omega_{n+1,n+1}) \;=\; -\frac{n+2}{2(n+1)}Z^2\kappa + Z - \frac{n+2}{2}\varepsilon Z\kappa + \frac{n(n+1)(n+2)}{24}\varepsilon^2\kappa.$$

We maximize $\mathcal{E}^{(1)}(A_n^{(\text{opt})}, \omega_{n+1,n+1})$ via the preceding expression by viewing the expectation as a function of Z. We discover that $\mathcal{E}^{(1)}(A_n^{(\text{opt})}, \omega_{n+1,n+1})$ is maximized when

$$Z \;=\; Z^{(\text{opt})} \;=\; \min\left\{\frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon,\; W\right\}. \tag{10}$$

For this case to be meaningful, the $(n+1)$th chunk must be nonempty, so that $A_n^{(\text{opt})} < Z$; i.e., $Z > \frac{1}{2}n(n+1)\varepsilon$. Therefore, we must simultaneously have:

1. $(n+1)/(n+2)X - \frac{1}{2}(n+1)\varepsilon > \frac{1}{2}n(n+1)\varepsilon$, so that $X > \frac{1}{2}(n+1)(n+2)\varepsilon$, which requires that $n \leq \left\lfloor \frac{1}{2}\left(\sqrt{1 + 8X/\varepsilon} - 3\right)\right\rfloor$.
2. $W > \frac{1}{2}n(n+1)\varepsilon$, which requires that $n \leq \left\lfloor \frac{1}{2}\left(\sqrt{1 + 8W/\varepsilon} - 1\right)\right\rfloor$.

We can now check the sanity of the result.

$$Z^{(\text{opt})} + (n+1)\varepsilon \;\leq\; \frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon + (n+1)\varepsilon \;<\; X,$$

because of the just established condition $\frac{1}{2}(n+1)(n+2)\varepsilon < X$. We also have,

$$A_n^{(\text{opt})} \;=\; \frac{n}{n+1}Z + \frac{n}{2}\varepsilon \leq \frac{n}{n+1}\left(\frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon\right) + \frac{n}{2}\varepsilon = \frac{n}{n+2}X < \frac{n}{n+1}X - \frac{n}{2}\varepsilon$$

because $\frac{1}{2}(n+1)(n+2)\varepsilon < X$. Therefore, the solution is consistent with the defining hypothesis for this case—namely, that $A_n < \frac{n}{n+1}X - \frac{n}{2}\varepsilon$.

Before moving on to case 2, we note that the value (9) does, indeed, extend our inductive hypothesis. To wit, the optimal total amount of allocated work, $Z^{(\text{opt})}$, has precisely the predicted value, and the sizes of the first n chunks do follow a decreasing arithmetic progression with common difference ε (by using the induction hypothesis). Finally, the last chunk has the claimed size:

$$\omega_{n+1,n+1} \;=\; Z^{(\text{opt})} - A_n^{(\text{opt})} \;=\; \frac{1}{n+1}Z^{(\text{opt})} - \frac{n}{2}\varepsilon.$$

We turn now to our remaining chores. We must derive the expectation-maximizing chunk sizes for the second case, wherein A_n is "big." And, we must show that the maximal expected work completion in this second case is always dominated by the solution of the first case—which will lead us to conclude that the regimen of the theorem is, indeed, optimal.

Case 2: $A_n \geq \dfrac{n}{n+1}X - \dfrac{n}{2}\varepsilon$.

By (7), if the current case's restriction on A_n is an *in*equality, then A_n *cannot* be an optimal cumulative n-chunk work allocation. We lose no generality, therefore, by focusing only on the subcase when the defining restriction of A_n is an *equality:*

$$A_n \;=\; \frac{n}{n+1}X - \frac{n}{2}\varepsilon.$$

For this value of A_n, call it $A_n^\star$, we have

$$\mathcal{E}_{\text{opt}}(W, n+1) \;=\; \max\Big(\mathcal{E}_{\text{opt}}(A_n^\star, n) + \omega_{n+1,n+1}\left(1 - (A_n^\star + \omega_{n+1,n+1} + (n+1)\varepsilon)\,\kappa\right)\Big),$$

where the maximization is over all values of $\omega_{n+1,n+1}$ in the closed interval $[0,\ W - A_n^\star]$.

To determine a value of $\omega_{n+1,n+1}$ that maximizes $\mathcal{E}_{\text{opt}}(W, n+1)$ for $A_n^\star$, we focus on the function

$$\begin{aligned}
\mathcal{E}^{(2)}(A_n, \omega_{n+1,n+1}) &= \mathcal{E}_{\text{opt}}(A_n, n) + \omega_{n+1,n+1}\left(1 - (A_n + \omega_{n+1,n+1} + (n+1)\varepsilon)\,\kappa\right) \\
&= \left(A_n - \frac{n+1}{2n}A_n^2\kappa - \frac{n+1}{2}A_n\varepsilon\kappa + \frac{(n-1)n(n+1)}{24}\varepsilon^2\kappa\right) \\
&\quad + \omega_{n+1,n+1}\left(1 - (A_n + \omega_{n+1,n+1} + (n+1)\varepsilon)\,\kappa\right) \\
&= -\kappa\omega_{n+1,n+1}^2 + \left(-\frac{n+2}{2}\kappa\varepsilon + \frac{1}{n+1}\right)\omega_{n+1,n+1} \\
&\quad + \frac{n\left((n+1)^2(n+2)\varepsilon^2\kappa^2 - 12(n+1)\varepsilon\kappa + 12\right)}{24(n+1)\kappa}.
\end{aligned}$$

Easily,

$$\frac{\partial\mathcal{E}^{(2)}(A_n, \omega_{n+1,n+1})}{\partial\omega_{n+1,n+1}} \;=\; -2\kappa\omega_{n+1,n+1} - \frac{n+2}{2}\kappa\varepsilon + \frac{1}{n+1}.$$

Knowing $A_n^\star$ exactly, we infer that the value of $\omega_{n+1,n+1}$ that maximizes $\mathcal{E}^{(2)}(A_n^\star, \omega_{n+1,n+1})$ is

$$\omega_{n+1,n+1} \;=\; \min\left\{\frac{1}{2(n+1)}X - \frac{1}{4}(n+2)\varepsilon,\ W - \frac{n}{n+1}X - \frac{n}{2}\varepsilon\right\}.$$

The second term dominates this minimization whenever

$$W \;\geq\; \frac{2n+1}{2n+2}X + \frac{n-2}{4}\varepsilon;$$

therefore, if W is large enough—as delimited by the preceding inequality—then

$$\mathcal{E}^{(2)}(A_n^\star, \omega_{n+1,n+1}) \;=\; \frac{2n^2+2n+1}{4(n+1)^2}X - \frac{2n^2+3n+2}{4(n+1)}\varepsilon + \frac{(n+2)(2n^2+5n+6}{48}\kappa\varepsilon^2,$$

When W does not achieve this threshold, then

$$\begin{aligned}\mathcal{E}^{(2)}(A_n^\star, \omega_{n+1,n+1}) = &- W^2\kappa + \left(\frac{n-2}{2}\kappa\varepsilon + \frac{2n+1}{n+1}\right)W \\ &+ \frac{(n^2+3n+14)n\kappa\varepsilon^2}{24} - \frac{n^2}{n+1}\varepsilon - \frac{n}{2(n+1)}X.\end{aligned}$$

For the found solution to be meaningful, the $(n+1)$th chunk must be nonempty, i.e., $\omega_{n+1,n+1} > 0$. This has two implications.

1. $X > \frac{(n+1)(n+2)}{2}\varepsilon$, which is true as long as $n \le \left\lfloor \frac{1}{2}\left(\sqrt{1+8X/\varepsilon} - 3\right)\right\rfloor$.
2. $W - (n/(n+1))X - \frac{1}{2}n\varepsilon > 0$, which implies $W > \frac{1}{2}n(n+1)\varepsilon$ because $X \ge W$. This inequality on W is true as long as $n \le \left\lfloor \frac{1}{2}\left(\sqrt{1+8W/\varepsilon} - 1\right)\right\rfloor$.

Because $X > \frac{1}{2}(n+1)(n+2)\varepsilon$, we have

$$A_n + \omega_{n+1,n+1} + (n+1)\varepsilon \;\le\; \frac{n}{n+1}X - \frac{n}{2}\varepsilon + \frac{1}{2(n+1)\kappa} - \frac{1}{4}(n+2)\varepsilon + (n+1)\varepsilon \;\le\; X.$$

For both Case 1 and Case 2, if either condition

$$\left[n \le \left\lfloor \frac{1}{2}\left(\sqrt{1+8X/\varepsilon} - 3\right)\right\rfloor\right] \quad \text{or} \quad \left[n \le \left\lfloor \frac{1}{2}\left(\sqrt{1+8W/\varepsilon} - 1\right)\right\rfloor\right]$$

does not hold, then there is no optimal schedule with $(n+1)$ nonempty chunks. (We will come back later to the case where one of these conditions does not hold.) If both conditions hold, then Case 1 always has an optimal schedule, but Case 2 may not have one.

To complete the proof, we must verify that the optimal regimen always corresponds to Case 1 (as suggested by the theorem), never to Case 2 (whenever Case 2 defines a valid solution). We accomplish this by considering two cases, depending on the size W of the workload. We show that the expected work completed under the regimen of Case 1 is never less than under the regimen of Case 2.

Case A: $W \ge \dfrac{n+1}{n+2}X - \dfrac{n+1}{2}\varepsilon$.

Under this hypothesis, and under Case 1, the workload that is actually deployed has size

$$Z \;=\; \frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon,$$

so that, in expectation,

$$\begin{aligned}\mathcal{E}^{(1)}(W, n+1) &= Z - \frac{n+1}{2n}Z^2\kappa - \frac{n+1}{2}Z\varepsilon\kappa + \frac{(n-1)n(n+1)}{24}\varepsilon^2\kappa \\ &= \frac{n+1}{2(n+2)}X - \frac{n+1}{2}\varepsilon + \frac{(n+1)(n+2)(n+3)}{24}\varepsilon^2\kappa.\end{aligned}$$

units of work are completed. Moreover, because

$$\frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon \;\leq\; \frac{2n+1}{2n+2}X + \frac{n-2}{4}\varepsilon,$$

the most favorable value for $\mathcal{E}^{(1)}(W, n+1)$ under Case 2 lies within the range of values for the current case. Because the value of $\mathcal{E}^{(1)}(W, n+1)$ is constant whenever

$$W \;\geq\; \frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon,$$

we can reach the desired conclusion by just showing that this value is no smaller than $\mathcal{E}^{(2)}\left(\frac{2n+1}{2n+2}X + \frac{n-2}{4}\varepsilon, n+1\right)$. Thus, we need only focus on the specific value

$$W_{\text{lim}} \;=\; \frac{2n+1}{2n+2}X + \frac{n-2}{4}\varepsilon$$

for W. For this value, we have:

$$\begin{aligned}\mathcal{E}^{(2)}(W_{\text{lim}}, n+1) &= -W_{\text{lim}}^2\kappa + \left(\frac{n-2}{2}\kappa\varepsilon + \frac{2n+1}{n+1}\right)W_{\text{lim}} + \frac{(n^2+3n+14)n\kappa\varepsilon^2}{24} \\ &\quad - \frac{n^2}{n+1}\varepsilon - \frac{n}{2(n+1)}X \\ &= \frac{2n^2+2n+1}{4(n+1)^2}X - \frac{2n^2+3n+2}{4(n+1)}\varepsilon + \frac{(n+2)(2n^2+5n+6)}{48}\varepsilon^2\kappa.\end{aligned}$$

By explicit calculation, we finally see that

$$\begin{aligned}\mathcal{E}^{(1)}(W, n+1) - \mathcal{E}^{(2)}(W_{\text{lim}}, n+1) &= \frac{\left(4 + (n^4+6n^3+13n^2+12n+4)\kappa^2\varepsilon^2\right)n}{16(n+1)^2(n+2)\kappa} \\ &\quad - \frac{(4n^2+12n+8)\kappa\varepsilon n}{16(n+1)^2(n+2)\kappa} \\ &= \frac{\left(4 + (n+1)^2(n+2)^2\kappa^2\varepsilon^2 - 4(n+1)(n+2)\kappa\varepsilon\right)n}{16(n+1)^2(n+2)\kappa} \\ &= \frac{\left((n+1)(n+2)\kappa\varepsilon - 2\right)^2 n}{16(n+1)^2(n+2)\kappa} \\ &\geq 0.\end{aligned}$$

Case B: $W \leq \frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon$.

In this case, the regimen of Case 1 deploys all W units of work, thereby completing, in expectation,

$$\mathcal{E}^{(1)}(W, n+1) = W - \frac{n+1}{2n}W^2\kappa - \frac{n+1}{2}W\varepsilon\kappa + \frac{(n-1)n(n+1)}{24}\varepsilon^2\kappa.$$

units of work. Moreover,

$$\frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon \;\leq\; \frac{2n+1}{2n+2}X + \frac{n-2}{4}\varepsilon,$$

so that the regimen of Case 2 also deploys all W units of work, thereby completing, in expectation,

$$\mathcal{E}^{(2)}(W, n+1) = -W^2\kappa + \left(\frac{n-2}{2}\kappa\varepsilon + \frac{2n+1}{n+1}\right)W + \frac{(n^2+3n+14)n\kappa\varepsilon^2}{24} - \frac{n^2}{n+1}\varepsilon - \frac{n}{2(n+1)}X.$$

units of work.

Explicit calculation now shows that

$$\mathcal{E}^{(1)}(W, n+1) - \mathcal{E}^{(2)}(W, n+1) = \frac{n}{2(n+1)}W^2\kappa - \frac{n}{n+1}(1+(n+1)\varepsilon\kappa)W + \frac{n}{2(n+1)}(1+2n\kappa\varepsilon-(n+1)\kappa^2\varepsilon^2)X.$$

Viewed as a function of W, this difference is, thus, unimodal, decreasing up to its global minimum, which occurs at $W = X + (n+1)\varepsilon$, and increasing thereafter. The largest value of W allowed by the current case is

$$W_{\max} \;=\; \frac{n+1}{n+2}X - \frac{n+1}{2}\varepsilon,$$

so this is also the value on which the difference $\mathcal{E}^{(1)}(W, n+1) - \mathcal{E}^{(2)}(W, n+1)$ reaches its minimum within its domain of validity. Thus, we need only focus on the behavior of the difference at the value $W = W_{\max}$. At this value,

$$\mathcal{E}^{(1)}(W_{\max}, n+1) - \mathcal{E}^{(2)}(W_{\max}, n+1) = \frac{n(5n+1)\varepsilon^2\kappa}{8} + \frac{(n-1)n\varepsilon}{2(n+1)(n+2)} + \frac{n}{2(n+1)(n+2)^2\kappa}.$$

This quantity is obviously positive, which means that $\mathcal{E}^{(1)}(W_{\max}, n+1) > \mathcal{E}^{(2)}(W_{\max}, n+1)$.

We thus see that, for workloads of any size W, one completes at least as much expected work via the schedule of Case 1 as via the schedule of Case 2.

In summation, if

$$n \;\le\; \min\left\{\left\lfloor\frac{1}{2}\left(\sqrt{1+8X/\varepsilon}-3\right)\right\rfloor,\; \left\lfloor\frac{1}{2}\left(\sqrt{1+8W/\varepsilon}-1\right)\right\rfloor\right\}, \tag{11}$$

then Case 1 specifies the optimal schedule that uses no more than $n+1$ chunks. Of course, this inequality translates to the conditions of the theorem (where it is written for n chunks instead of $n+1$).

Note that if n exceeds either quantity in the minimization of (11), then one never improves the expected amount of work completed by deploying the workload in more than n chunks. This is another consequence of our remark about A_n at the beginning of this proof. If there exists a value of m for which there exists a schedule $\mathcal{S}$ that uses $\ge n+1$ nonempty chunks, then replacing the first $n+1$ chunks in this solution with the optimal solution for n chunks, using a workload equal to the first $n+1$ chunks of $\mathcal{S}$, yields a schedule that, in expectation, completes strictly more work than $\mathcal{S}$. □

4 Scheduling for Two Remote Computers

Before we approach the general case of p remote computers, we study the case of *two* remote computers, in order to adduce principles that will be useful in the general case. We first establish characteristics of optimal schedules under general risk functions, then restrict attention to the linear risk model. 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

Focus on a distribution of work to P_1 and P_2 under which, for $i = 1, 2$, P_i receives n_i chunks to execute, call them $\mathcal{W}_{i,1}, \ldots, \mathcal{W}_{i,n_i}$, to be scheduled in this order; as usual we denote $|\mathcal{W}_{i,j}|$ by $\omega_{i,j}$. We do not assume any *a priori* relation between the way P_1 and P_2 break their allocated work into chunks; in particular, any work that is allocated to both P_1 and P_2 may be chunked quite differently on the two machines.

Theorem 4. (Two remote computers: free-initiation model; general risk)
Let Θ be a schedule for two remote computers, P_1 and P_2. Say that, for both P_1 and P_2, the probability of being interrupted never decreases as a computer processes more work. There exists a schedule Θ' for P_1 and P_2 that, in expectation, completes as much work as does Θ and that satisfies the following three properties; cf. Fig. 1.

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

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.*

Shared work "mirroring." *Θ' has P_1 and P_2 process their shared work "in opposite orders." Specifically, say that P_1 chops its allocated work into chunks $\mathcal{W}_{1,1}, \ldots, \mathcal{W}_{1,n_1}$, while P_2 chops its allocated work into chunks $\mathcal{W}_{2,1}, \ldots, \mathcal{W}_{2,n_2}$.*

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: chunks $\mathcal{W}_{1,a_1}$ and $\mathcal{W}_{2,a_2}$ both contain a shared "piece of work" A, and chunks $\mathcal{W}_{1,b_1}$ 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 chunk $\mathcal{W}_{1,a_1}$ before chunk $\mathcal{W}_{1,b_1}$), then Θ' has P_2 execute B before A (i.e., P_2 executes chunk $\mathcal{W}_{2,b_2}$ before chunk $\mathcal{W}_{2,a_2}$).

Proof. We need to refine our description of workloads. We describe the overall workload of size W via a partition $\mathcal{X} = \{\mathcal{X}_1, ..., \mathcal{X}_m\}$ such that, for $j = 1, 2$, each element $\mathcal{X}_i \in \mathcal{X}$ is either included in a chunk of P_j or does not intersect any chunk of P_j. Symbolically:

$$(\forall i \in [1..m]) \left[\mathcal{X}_i \cap \bigcup_{k=1}^{n_1} \mathcal{W}_{j,k} = \emptyset \right] \quad \text{OR} \quad (\exists k \in [1..n_1]) \left[\mathcal{X}_i \subset \mathcal{W}_{j,k} \right].$$

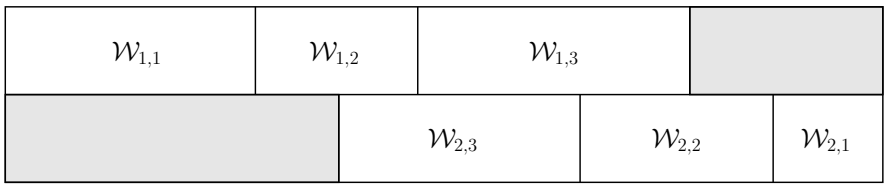


Figure 1: The shape of an optimal schedule for two computers, as described in Theorem 4; $n_1 = n_2 = 3$. The top row displays P_1's chunks, the bottom row P_2's. Vertically aligned parts of chunks correspond to shared work; shaded areas depict unallocated work (e.g., none of the work in $\mathcal{W}_{2,1}$ is allocated to P_1).

We then define, for $j = 1, 2$ and $i = 1, \ldots n_j$,

$$\delta_j(i) \;=\; \begin{cases} 0 & \text{just when element } \mathcal{X}_i \in \mathcal{X} \text{ does not intersect any chunk of } P_j \\ 1 & \text{otherwise.} \end{cases}$$

When $\delta_j(i) = 1$, we denote by $\sigma_j(i)$ the chunk of P_j that contains $\mathcal{X}_i$, so that $\mathcal{X}_i \subset \mathcal{W}_{j,\sigma_j(i)}$.

We can now specify the expectation $\mathcal{E}$ of jobdone under the described distribution of chunks to P_1 and P_2. The probability that element $\mathcal{X}_i \in \mathcal{X}$ is computed successfully is:

- 0 if $\mathcal{X}_i$ is not allocated to either P_1 or P_2;
- $1 - Pr\left(\sum_{j=1}^{\sigma_1(i)} \omega_{1,j}\right)$ if $\mathcal{X}_i$ is allocated to P_1 but not to P_2, i.e., if $\delta_1(i)(1 - \delta_2(i)) = 1$;
- $1 - Pr\left(\sum_{j=1}^{\sigma_2(i)} \omega_{2,j}\right)$ if $\mathcal{X}_i$ is allocated to P_2 but not to P_1, i.e., if $(1 - \delta_1(i))\delta_2(i) = 1$;
- $1 - Pr\left(\sum_{j=1}^{\sigma_1(i)} \omega_{1,j}\right) Pr\left(\sum_{j=1}^{\sigma_2(i)} \omega_{2,j}\right)$ if $\mathcal{X}_i$ is allocated to both P_1 and P_2, i.e., if $\delta_1(i)\delta_2(i) = 1$.

We thus end up with the following overall expression for $\mathcal{E}$, the expected amount of work completed under the described distribution:

$$\mathcal{E} = \sum_{i=1}^{m} |\mathcal{X}_i| \cdot \Xi_i,$$

where

$$\begin{aligned} \Xi_i \;=\; & \delta_1(i)\delta_2(i) \left(1 - Pr\left(\sum_{j=1}^{\sigma_1(i)} \omega_{1,j}\right) Pr\left(\sum_{j=1}^{\sigma_2(i)} \omega_{2,j}\right)\right) \\ & + \delta_1(i)(1 - \delta_2(i)) \left(1 - Pr\left(\sum_{j=1}^{\sigma_1(i)} \omega_{1,j}\right)\right) \\ & + (1 - \delta_1(i))\delta_2(i) \left(1 - Pr\left(\sum_{j=1}^{\sigma_2(i)} \omega_{2,j}\right)\right). \end{aligned}$$

We now verify the existence of an optimal schedule Θ that satisfies each of the three claimed properties. In fact, we prove a slightly stronger claim, by verifying these properties of Θ for the partition elements rather than for the chunks.

Maximal work deployment. Say, for contradiction, that some piece of the overall workload is allocated to both P_1 and P_2, while another piece is allocated to neither. Since we can arbitrarily subdivide the partition $\mathcal{X}$, we may assume with no loss of generality that

- the doubly allocated piece is a *partition element* $\mathcal{X}_i$;
- the unallocated piece is a *partition element* $\mathcal{X}_j$ *of the same size* as $\mathcal{X}_i$.

We construct a new allocation by substituting $\mathcal{X}_j$ for $\mathcal{X}_i$ in $\mathcal{W}_{1,\sigma_1(i)}$; in the overall expectation, only the terms involving $\mathcal{X}_i$ and $\mathcal{X}_j$ are affected by this substitution. Now, before the substitution, the total contribution of these terms to the overall expectation was:

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right) + |\mathcal{X}_j| \times 0.$$

After the substitution, this contribution becomes:

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right) + |\mathcal{X}_j| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right)\right).$$

Because $|\mathcal{X}_i| = |\mathcal{X}_j|$, the latter contribution is never less than the former, differing from it by the nonnegative quantity

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) - Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right) + Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right).$$

$$= |\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right)\right)\left(1 - Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right).$$

We have thus verified that there is always an optimal schedule that honors the first claimed property. We shall henceforth restrict attention to such schedules.

Local work priority. Assume next, for contradiction, that under the optimal schedule Θ, there are two elements $\mathcal{X}_i, \mathcal{X}_j \in \mathcal{X}$ such that:

1. $\mathcal{X}_i$ is allocated to P_1 but not to P_2: symbolically, $\delta_1(i)(1 - \delta_2(i)) = 1$;
2. $\mathcal{X}_j$ is allocated to both P_1 and P_2: symbolically, $\delta_1(i)\delta_2(i) = 1$;
3. Schedule Θ attempts to execute $\mathcal{X}_i$ after $\mathcal{X}_j$ on P_1: symbolically, $\sigma_1(i) \geq \sigma_1(j)$.

As before, we may assume, with no loss of generality, that $|\mathcal{X}_i| = |\mathcal{X}_j|$. We now construct a new schedule for P_1 by substituting $\mathcal{X}_j$ for $\mathcal{X}_i$ in $\mathcal{W}_{1,\sigma_1(i)}$ and substituting $\mathcal{X}_i$ for $\mathcal{X}_j$ in $\mathcal{W}_{1,\sigma_1(j)}$, while leaving the schedule of P_2 unchanged. Once again, these substitutions affect

only the terms involving $\mathcal{X}_i$ and $\mathcal{X}_j$ in the overall expectation. The total contribution of these terms before the substitution was:

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right)\right) + |\mathcal{X}_j| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(j)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(j)} \omega_{2,k}\right)\right).$$

After the substitution, the contribution becomes

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(j)} \omega_{1,k}\right)\right) + |\mathcal{X}_j| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(j)} \omega_{2,k}\right)\right).$$

Because $|\mathcal{X}_i| = |\mathcal{X}_j|$, we see that the substitution increases the overall expectation by the quantity

$$|\mathcal{X}_i| \times \left(Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) - Pr\left(\sum_{k=1}^{\sigma_1(j)} \omega_{1,k}\right)\right) \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_2(j)} \omega_{2,k}\right)\right),$$

This quantity is nonnegative because

- the probability $Pr(w)$ is nondecreasing in w;
- $\sum_{k=1}^{\sigma_1(i)} \omega_{1,k} \geq \sum_{k=1}^{\sigma_1(j)} \omega_{1,k}$, because $\sigma_1(i) \geq \sigma_1(j)$.

The modified schedule completes, in expectation, at least as much work as Θ, and it satisfies the second property.

Shared work "mirroring." Finally, consider a schedule under which there are two partition elements, $\mathcal{X}_i$ and $\mathcal{X}_j$, such that:

1. Both $\mathcal{X}_i$ and $\mathcal{X}_j$ are allocated to both P_1 and P_2: symbolically, $\delta_1(i)\delta_2(i) = \delta_1(j)\delta_2(j) = 1$;
2. The optimal schedule Θ attempts to execute $\mathcal{X}_i$ after $\mathcal{X}_j$ on both P_1 and P_2: symbolically, $[\sigma_1(i) \geq \sigma_1(j)]$ and $[\sigma_2(i) \geq \sigma_2(j)]$.

As before, we may assume with no loss of generality that the $|\mathcal{X}_i| = |\mathcal{X}_j|$. We now construct a new schedule by substituting $\mathcal{X}_j$ for $\mathcal{X}_i$ in $\mathcal{W}_{1,\sigma_1(i)}$ and substituting $\mathcal{X}_i$ to $\mathcal{X}_j$ in $\mathcal{W}_{1,\sigma_1(j)}$, leaving the schedule of P_2 unchanged. Once again, these changes affect only the terms involving $\mathcal{X}_i$ and $\mathcal{X}_j$ in the overall expectation. The total contribution of these terms before the substitution was:

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right) + |\mathcal{X}_j| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(j)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(j)} \omega_{2,k}\right)\right).$$

After the substitution, their contribution becomes:

$$|\mathcal{X}_i| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(j)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right) + |\mathcal{X}_j| \times \left(1 - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right) Pr\left(\sum_{k=1}^{\sigma_2(j)} \omega_{2,k}\right)\right).$$

Because $|\mathcal{X}_i| = |\mathcal{X}_j|$, the substitutions have increased the overall expectation by the quantity:

$$|\mathcal{X}_i| \times \left(Pr\left(\sum_{k=1}^{\sigma_1(j)} \omega_{1,k}\right) - Pr\left(\sum_{k=1}^{\sigma_1(i)} \omega_{1,k}\right)\right)\left(Pr\left(\sum_{k=1}^{\sigma_2(j)} \omega_{2,k}\right) - Pr\left(\sum_{k=1}^{\sigma_2(i)} \omega_{2,k}\right)\right).$$

This quantity is nonnegative because $\mathcal{X}_i$ and $\mathcal{X}_j$ are processed in the same order on P_1 and on P_2. The modified schedule thus completes, in expectation, at least as much work as Θ, and it satisfies the third property. □

4.2 Two Remote Computers under Linear Risk

4.2.1 Allocating work in a single chunk

We now specialize from general risk functions to the linear risk functions. We first consider the case wherein each computer processes its allocated work as a single chunk. Even this simple case turns out to be surprisingly difficult to schedule optimally when there is more than one remote computer. Indeed, our experience with this case leads us to abandon the quest for exactly optimal schedules, in favor of the more easily accessed *asymptotically* optimal schedules.

> "Asymptotically optimal" here means that the expected amount of work completed by these schedules deviates from exact maximality by an amount that shrinks as the size of the workload grows without bound.

To render the single-chunk scheduling problem tractable, we restrict attention to schedules that are *symmetric,* in the sense that they allocate the same amount of work to each remote computer. It seems intuitive that there is always a symmetric schedule among the optimal single-chunk schedules, but we have yet to verify this.

Say that our workload consists of W units of work that we somehow order linearly. We denote by $\langle a, b\rangle$ the sub-workload obtained by eliminating: the initial a units of work and all work beyond the initial b units. For instance, $\langle 0, W\rangle$ denotes the entire workload, $\langle 0, \frac{1}{2}W\rangle$ denotes the first half of the workload, and $\langle \frac{1}{2}W, W\rangle$ denotes the last half of the workload.

Theorem 5. (Two remote computers: linear risk; single chunk allocation) *Say that we wish to deploy W units of work on two computers, deploying a single chunk per computer. The following symmetric schedule Θ completes, in expectation, a maximum amount of work.*

- *If $W \leq \frac{1}{2}X$, then Θ deploys the entire workload on both remote computers; symbolically, $\mathcal{W}_{1,1} = \mathcal{W}_{2,1} = \langle 0, W\rangle$;*
- *if $\frac{1}{2}X < W \leq X$, then Θ deploys the first half of the workload on one computer and the second half on the other; symbolically, $\mathcal{W}_{1,1} = \langle 0, \frac{1}{2}W\rangle$, and $\mathcal{W}_{2,1} = \langle \frac{1}{2}W, W\rangle$;*
- *if $X < W$, then Θ deploys only X units of the workload, allocating the first half to one computer and the second half to the other; symbolically, $\mathcal{W}_{1,1} = \langle 0, \frac{1}{2}X\rangle$, and $\mathcal{W}_{2,1} = \langle \frac{1}{2}X, X\rangle$.*

Proof. Our derivation of the optimal schedule builds on the following principle (which we have encountered before). When we deploy work as a single chunk, we never make that chunk as large as X, for then we risk certain interruption, hence, in expectation, completes no work. In order to see how to optimally deploy work as a single chunk, we consider separately schedules that allow overlapping deployments to the two computers and those that do not.

Disjoint allocations. Focus first on schedules that deploy non-overlapping workloads to the two remote computers. These two workloads, $\mathcal{W}_{1,1}$ and $\mathcal{W}_{2,1}$, are independent, so we can invoke Theorem 2 to discover their optimal sizes. We see that the optimal strategy is to deploy $Z = \min\{W, X\}$ units of work in total. We determine the optimal allocation of this work to the remote computers, in respective chunk sizes $\omega_{1,1}$ and $\omega_{2,1} = Z - \omega_{1,1}$, by considering the expectation of jobdone.

$$\begin{aligned}\mathcal{E} &= \omega_{1,1}\left((1-\omega_{1,1}\kappa) + (Z-\omega_{1,1})(1-(Z-\omega_{1,1})\kappa)\right)\\ &= -2\omega_{1,1}^2\kappa + 2\omega_{1,1}Z\kappa + Z - Z^2\kappa.,\end{aligned}$$

Easily, $\mathcal{E}$ is maximized when $\omega_{1,1} = \omega_{2,1} = \frac{1}{2}Z$. The optimal value of $\mathcal{E}$ is, then, $\mathcal{E} = W - \frac{1}{2}Z^2\kappa$. (Note that we did not need to *assume* that the optimal schedule is symmetric in this case; we actually proved that it should be.)

Overlapping allocations to the two computers. The principle enunciated at the beginning of this proof implies that an optimal schedule that deploys overlapping workloads to the two remote computers never allocates a full X units of work to either computer. We can, therefore, simplify our calculations by restricting attention henceforth to the case $W < 2X$. Since we consider only symmetric schedules, the common size s of the allocations to both computers satisfies $s \geq \frac{W}{2}$, by Theorem 4. We thus obtain the following expression for the expectation of jobdone as a function of s.

$$\begin{aligned}\mathcal{E}(s) &= 2(W-s)(1-s\kappa) + (2s-W)(1-s^2\kappa^2)\\ &= -2s^3\kappa^2 + (2+W\kappa)s^2\kappa - 2sW\kappa + W.\end{aligned}$$

We seek the maximizing value of s.

$$\mathcal{E}'(s) = \frac{\mathrm{d}}{\mathrm{d}s}\mathcal{E}(s) = 2[-3s^2\kappa + (2+W\kappa)s - W]\kappa.$$

The discriminant of the bracketed quadratic polynomial is

$$\Delta = W^2\kappa^2 - 8W\kappa + 4 = (W\kappa - 2(2+\sqrt{3}))(W\kappa - 2(2-\sqrt{3})).$$

Because $W < 2X$ we have, $W\kappa < 2(2+\sqrt{3})$. We branch on the relative sizes of W and $2(2-\sqrt{3})X$:

$\underline{W > 2(2-\sqrt{3})X}$. In this case, $\Delta < 0$, so the polynomial has no real roots, and $\mathcal{E}(s)$ is decreasing with s. Because $s \in [\frac{1}{2}W, W]$, $\mathcal{E}(s)$ is maximized when $s = W/2$.

$\underline{W \leq 2(2-\sqrt{3})X}$. This case is far more complicated than its predecessor. Let us denote the two roots of our quadratic polynomial by s_- and s_+, as follows:

$$s_- = \frac{2 + W\kappa - \sqrt{W^2\kappa^2 - 8W\kappa + 4}}{6\kappa} \quad \text{and} \quad s_+ = \frac{2 + W\kappa + \sqrt{W^2\kappa^2 - 8W\kappa + 4}}{6\kappa}.$$

One sees that $\mathcal{E}'(s)$ decreases as s progresses from $-\infty$ to s_-, then decreases as s progresses from s_- to s_+, then decreases once more as s increases beyond s_+. We must determine how these three intervals overlap $\mathcal{E}$'s domain of validity, viz., $s \in [\frac{1}{2}W,\ W]$.

We note first that $\frac{1}{2}W \leq s_-$. Indeed:

$$\begin{array}{lll} & W/2 & \geq \quad s_- \\ \text{just when} & W/2 & \geq \quad \dfrac{2 + W\kappa - \sqrt{W^2\kappa^2 - 8W\kappa + 4}}{6\kappa} \\ \\ \text{just when} & \sqrt{W^2\kappa^2 - 8W\kappa + 4} & \geq \quad 2(1 - W\kappa) \\ \text{only if} & 0 & \geq \quad 3W^2\kappa^2 \end{array}$$

We invoke here the fact that $W\kappa \leq 1$, because $W \leq 2(2-\sqrt{3})X \leq X$. We remark next that $\mathcal{E}'(W) = 2W\kappa(1 - 2W\kappa)$, so that $\mathcal{E}'(W) \geq 0$ and $W \in [s_-, s_+]$ when $W \leq \frac{X}{2}$; moreover, $W > s_+$ when $W > \frac{1}{2}X$. Indeed, if we assume that $\frac{1}{2}X < W \leq s_+$ (the lower bound implying $5W\kappa - 2 \geq 0$), then we reach a contradiction:

$$\begin{array}{lll} & W & \leq \quad s_+ \\ \text{just when} & W & \leq \quad \dfrac{2 + W\kappa + \sqrt{W^2\kappa^2 - 8W\kappa + 4}}{6\kappa} \\ \\ \text{just when} & 5W\kappa - 2 & \leq \quad \sqrt{W^2\kappa^2 - 8W\kappa + 4} \\ \text{only if} & 2W\kappa & \leq \quad 1. \end{array}$$

So, once again we have two cases to consider:

$\underline{W \leq X/2}$. In this case, we have $W \in [s_-, s_+]$, so that $\mathcal{E}(s)$ achieves its maximum either when $s = \frac{1}{2}W$ or when $s = W$. Hence $\mathcal{E}(s)$'s maximum is either

$$\mathcal{E}(W/2) \ = \ W - \frac{1}{2}W^2\kappa \qquad \text{or} \qquad \mathcal{E}(W) \ = \ W - W^3\kappa^2.$$

When $W \leq \frac{1}{2}X$, which is the case here, the latter value dominates, so the optimal deployment is $\omega_{1,1} = \omega_{2,1} = W$.

$\underline{W > X/2}$. In this case, $W > s_+$, so that $\mathcal{E}(s)$ achieves its maximum either when $s = W/2$ or when $s = s_+$. We compare the values at these points by computing both $\mathcal{E}(s_+)$ and $\mathcal{E}(s_+) - \mathcal{E}(W/2)$. We find that

$$\mathcal{E}(s_+) \ = \ \frac{(W^2\kappa^2 - 8W\kappa + 4)^{\frac{3}{2}} + W^3\kappa^3 - 12W^2\kappa^2 + 30W\kappa + 8}{54\kappa}$$

and

$$\mathcal{E}(s_+) - \mathcal{E}(W/2) \ = \ \frac{[W^2\kappa^2 - 8W\kappa + 4]^{3/2} + [W^3\kappa^3 + 15W^2\kappa^2 - 24W\kappa + 8]}{54\kappa}. \qquad (12)$$

Easily, both of the bracketed polynomials in (12) *decrease* as W progresses along its current hypothesized interval, from $\frac{1}{2}X$ through $2(2-\sqrt{3})X$; therefore, the difference $\mathcal{E}(s_+) - \mathcal{E}(W/2)$ decreases as W proceeds along the same interval. Since the difference vanishes at the point $W = \frac{1}{2}X$, we conclude that the optimal deployment in this case is $\omega_{1,1} = \omega_{2,1} = \frac{1}{2}W$. □

Moving beyond Theorem 5. The preceding analysis determines the optimal schedule for only two remote computers that each process their allocations as single chunks. The complexity of even this simple case has led us to abandon our focus on exactly optimal schedules in the sequel, in favor of a hopefully more tractable search for schedules that are asymptotically optimal.

> Since one can view schedules under the free-initiation model as "asymptotic versions" of schedules under the charged-initiation model—cf. Theorem 1—our shift in focus is not a drastic one.

This shift in focus notwithstanding, 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,

$$\mathcal{E} = W - \frac{W^3\kappa^2}{6}\left(1 + \frac{3}{n} + \frac{2}{n^2}\right)$$

units of work (cf. Theorem 6 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 Figs. 2 and 3 contain chunk sizes. In Fig. 2, for instance, each computer uses m chunks of size α, two chunks of size $\frac{1}{4}(W - m\alpha)$, and one chunk of size $\frac{1}{2}(W - m\alpha)$.

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.

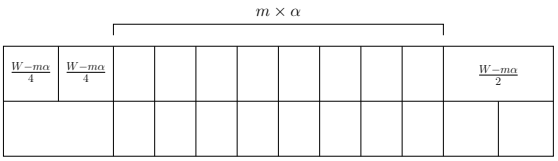


Figure 2: Counterexample to the optimality of schedules that employ equal-size chunks.

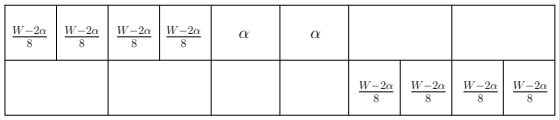


Figure 3: Counterexample to the optimality of the schedule of Fig. 2.

4.2.2 Asymptotically optimal schedules

This section is devoted to Algorithm 1, whose prescribed schedules for two remote computers branch on the value of $W\kappa$. We show in Theorem 6 that the proposed schedules are all asymptotically optimal; they are exactly optimal when $W\kappa \geq 2$.

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

if $W \geq 2X$ **then**
- $\forall i \in [1,n],\ \mathcal{W}_{1,i} \leftarrow \left\langle \frac{i-1}{n} \cdot \frac{n}{n+1} X,\ \frac{i}{n} \cdot \frac{n}{n+1} X \right\rangle$
- $\forall i \in [1,n],\ \mathcal{W}_{2,i} \leftarrow \left\langle W - \frac{i}{n} \cdot \frac{n}{n+1} X,\ W - \frac{i-1}{n} \cdot \frac{n}{n+1} X \right\rangle$

if $W \leq X$ **then**
- $\forall i \in [1,n],\ \mathcal{W}_{1,i} = \mathcal{W}_{2,n-i+1} \leftarrow \left\langle \frac{i-1}{n} W,\ \frac{i}{n} W \right\rangle$

if $X < W < 2X$ **then**
- $\ell \leftarrow \lfloor n/3 \rfloor$
- $\forall i \in [1,\ell],\ \mathcal{W}_{1,i} \leftarrow \left\langle \frac{i-1}{\ell}(W-X),\ \frac{i}{\ell}(W-X) \right\rangle$
- $\forall i \in [1,\ell],\ \mathcal{W}_{2,i} \leftarrow \left\langle W - \frac{i}{\ell}(W-X),\ W - \frac{i-1}{\ell}(W-X) \right\rangle$
- $\forall i \in [1,2l],$ $\mathcal{W}_{1,l+i} = \mathcal{W}_{2,3l-i+1} \leftarrow \left\langle (W-X) + \frac{i-1}{2\ell}(2X-W),\ (W-X) + \frac{i}{2\ell}(2X-W) \right\rangle$

Theorem 6. *The schedules specified by Algorithm 1 are:*

1. *optimal when $W \geq 2X$; in expectation, they complete*

$$E^{(\mathrm{f},2)}(W, \textit{Algorithm 1}(n)) = \frac{n-1}{n} X$$

units of work, which tends to[8] X*;*

2. *asymptotically optimal when $W \leq X$; in expectation, they complete*

$$E^{(\mathrm{f},2)}(W, \textit{Algorithm 1}(n)) = W - \frac{1}{6} W^3 \kappa^2 \left(1 + \frac{3}{n} + \frac{2}{n^2}\right)$$

[8] "tends to" means "as n grows without bound."

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*

$$E^{(\mathrm{f},2)}(W, \textit{Algorithm 1}(n)) = 2W - \frac{1}{3}X - W^2\kappa + \frac{1}{6}W^3\kappa^2 + \frac{1}{\ell}\left(\left(1+\frac{1}{\ell}\right)W - \left(1+\frac{2}{3\ell}\right)X - \frac{1}{2\ell}W^2\kappa - \frac{1}{4}\left(1-\frac{1}{3\ell}\right)W^3\kappa^2\right)$$

units of work, which tends to $2W - \frac{1}{3}X - W^2\kappa + \frac{1}{6}W^3\kappa^2$*.*

Proof. We prove the theorem's three assertions in turn.

1. Case: $W \geq 2X$.

   By definition of X, a computer is certain to be interrupted when processing a workload of size at least X. Therefore, when $W \geq 2X$, Theorem 4 tells us that the two computers are working on disjoint subsets of the workload. Then Theorem 2 defines the sizes of these workloads and the way they are partitioned into chunks. Theorem 2 also gives us the expectation of jobdone.

2. Case: $W \leq X$.

   We must prove two things: that the proposed schedule is asymptotically optimal and that its expectation for jobdone is what we claim it is. Let us take any positive integer n. Then, the expectation of jobdone under Algorithm 1 is no greater than this expectation under the optimal scheduling:

   $$E^{(\mathrm{f},2)}(W, n) \geq E^{(\mathrm{f},2)}(W, \text{Algorithm } 1(n)).$$

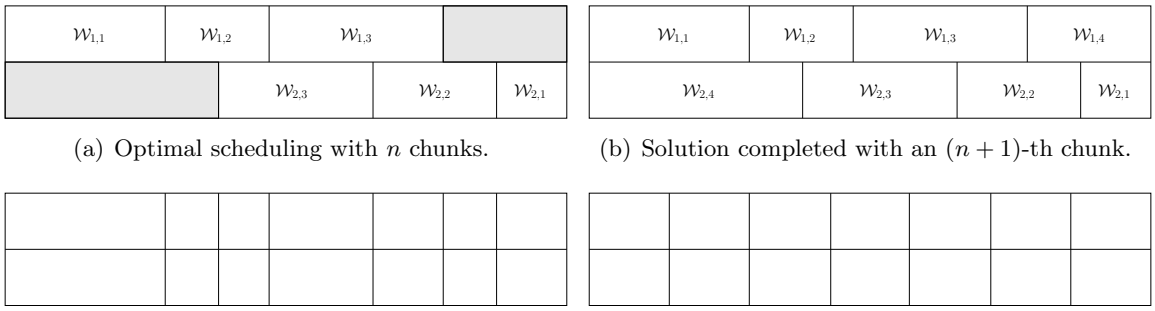
   Following the series of transformations illustrated by Figure 4, we show that the optimal scheduling with n chunks has not a better expectation than the solution of Algorithm 1 with $2n+1$ chunks. Each transformation is a non-decreasing transformation from the point of view of the expectation of jobdone.

   We start from an optimal scheduling for n chunks satisfying Theorem 4 (see Figure 4(a)). First, we add a possibly empty $(n+1)$-th chunk to the workload of each computer so that each computer processes the whole workload (see Figure 4(b)). Obviously, this transformation does not decrease the expectation. Then, we subdivide the chunks so that the boundaries of the chunks of a computer coincide with the boundaries of the chunks of the other computer (see Figure 4(c)). Formally, let $\mathcal{B}_1$ (resp. $\mathcal{B}_2$) be the set of the "places" in the workload at which there is the boundary of a chunk of computer 1 (resp. of computer 2): $\mathcal{B}_1 = \bigcup_{i=0}^{n+1} \left\{\sum_{j=1}^{i} \omega_{1,j}\right\}$ (resp. $\mathcal{B}_2 = \bigcup_{i=0}^{n+1} \left\{W - \sum_{j=1}^{i} \omega_{2,j}\right\}$). Then, we take the union of these two sets, and we order its elements:

   $$\mathcal{B}_1 \cup \mathcal{B}_2 = \{b_1, ..., b_l\} \text{ with } 0 = b_0 < b_1 < b_2 < ... < b_{l-1} < b_l = W.$$

   Finally, the new chunks are defined by partitioning the whole workload into l chunks such that:

   $$\mathcal{W}'_{1,i} = \mathcal{W}'_{2,l-i+1} \text{ with } \omega'_{1,i} = b_i - b_{i-1}.$$



(a) Optimal scheduling with n chunks.

(b) Solution completed with an $(n+1)$-th chunk.

(c) Dividing chunks for chunk boundaries to coincide.

(d) Solution of Algorithm 1 with $2n+1$ chunks.

Figure 4: Series of schedule transformations to prove the asymptotic optimality of Algorithm 1 when $W \leq X$.

We then remark that l is no greater than $2n+1$: $l \leq 2n+1$. Indeed, there are at most n chunks boundaries on computer 1 (resp. on computer 2) which are strictly between 0 and W. Therefore, in the new schedule, there are at most $2n$ chunk boundaries strictly between 0 and W, which defines at most $2n+1$ chunks with the boundaries of the whole workload. Finally, we remark that subdividing chunks does not decrease the overall expectation.

To move closer to the schedule produced by Algorithm 1, we replaced the l chunks we just built, by l chunks of equal-sizes: $\forall i \in [1,l], \mathcal{W}''_{1,i} = \mathcal{W}''_{2,l-i+1} = [(i-1)\frac{W}{l}, i\frac{W}{l}]$. To prove that this indeed gives us a better solution, we prove a more general result.

Let us consider a scheduling over two computers. Computer P_1 executes the chunks $\mathcal{V}_{1,1}$, ..., $\mathcal{V}_{1,l_1}$ in that order, and Computer P_2 executes the chunks $\mathcal{V}_{2,1}$, ..., $\mathcal{V}_{2,l_2}$ in that order. Moreover, we suppose that the two computers have two consecutive chunks in common. More formally, we assume that there exists two indexes $i \in [1, l_1 - 1]$ and $j \in [1, l_2 - 1]$ such that $\mathcal{V}_{1,i} = \mathcal{V}_{2,j+1}$ and $\mathcal{V}_{1,i+1} = \mathcal{V}_{2,j}$ [9]. We look at how best distribute the load between $\mathcal{V}_{1,i}$ ($=\mathcal{V}_{2,j+1}$) and $\mathcal{V}_{1,i+1}$ ($=\mathcal{V}_{2,j}$). So, we let the scheduling unchanged except for the distribution of the load $|\mathcal{V}_{1,i}|+|\mathcal{V}_{1,i+1}|$ between chunks $\mathcal{V}_{1,i}$ and $\mathcal{V}_{1,i+1}$. Therefore, we only focus on the contribution of these two chunks to the overall expectation:

$$\mathcal{E} = |\mathcal{V}_{1,i}| \left(1 - \left(\sum_{k=1}^{i} |\mathcal{V}_{1,k}|\kappa\right)\left(\sum_{k=1}^{j+1} |\mathcal{V}_{2,k}|\kappa\right)\right) + |\mathcal{V}_{1,i+1}| \left(1 - \left(\sum_{k=1}^{i+1} |\mathcal{V}_{1,k}|\kappa\right)\left(\sum_{k=1}^{j} |\mathcal{V}_{2,k}|\kappa\right)\right).$$

To simplify the writing of formulas, we use the following notations: $V_1 = \sum_{k=1}^{i-1} |\mathcal{V}_{1,k}|$, $V_2 = \sum_{k=1}^{j-1} |\mathcal{V}_{2,k}|$, and $L = |\mathcal{V}_{1,i}| + |\mathcal{V}_{1,i+1}|$. Then,

$$\mathcal{E} = |\mathcal{V}_{1,i}| \left(1 - (V_1 + |\mathcal{V}_{1,i}|)\kappa(V_2 + L)\kappa\right) + (L - |\mathcal{V}_{1,i}|) \left(1 - (V_1 + L)\kappa(V_2 + L - |\mathcal{V}_{1,i}|)\kappa\right).$$

Therefore, the contribution of these two chunks to the expectation is equal to:

$$\mathcal{E} = -(V_1 + V_2 + 2L)\kappa^2 |\mathcal{V}_{1,i}|^2 + (V_1 + V_2 + 2L)L\kappa^2 |\mathcal{V}_{1,i}| + L(1 - (V_1 + L)(V_2 + L)\kappa^2)$$

and this expression is maximized when $|\mathcal{V}_{1,i}| = \frac{L}{2}$, that is, when both chunks have the same size.

[9] Theorem 4 tells us that the case $\mathcal{V}_{1,i} = \mathcal{V}_{2,j}$ and $\mathcal{V}_{1,i+1} = \mathcal{V}_{2,j+1}$ is suboptimal.

So replacing l coinciding chunks by l equal-size chunks does not decrease the expectation. Finally, to obtain a well defined bound using the schedule of Algorithm 1, we just enlarge the number of (equal-size) chunks, going from l to $2n+1$. To prove that this last transformation do not decrease the overall expectation, we just explicit the expectation of a solution with n equal-size chunks per computer and we show that this expectation is non-decreasing with n.

$$E^{(\mathrm{f},2)}(W,n) = \sum_{i=1}^{n} \frac{W}{n}\left(1 - \sum_{j=1}^{i} \frac{W}{n}\kappa \sum_{j=1}^{n-i+1} \frac{W}{n}\kappa\right).$$

$$\begin{aligned} E^{(\mathrm{f},2)}(W,n) &= \sum_{i=1}^{n} \frac{W}{n}\left(1 - \sum_{j=1}^{i} \frac{W}{n}\kappa \sum_{j=1}^{n-i+1} \frac{W}{n}\kappa\right) \\ &= W - \frac{W^3}{n^3}\kappa^2 \sum_{i=1}^{n}(i(n+1-i)) \\ &= W - \frac{W^3\kappa^2}{6}\left(1 + \frac{3}{n} + \frac{2}{n^2}\right). \end{aligned}$$

The last expression is obviously a sum of non-decreasing functions in n.

We have therefore proved that for any positive value of n:

$$E^{(\mathrm{f},2)}(W, \text{Algorithm 1}(2n+1)) \geq E^{(\mathrm{f},2)}(W,n) \geq E^{(\mathrm{f},2)}(W, \text{Algorithm 1}(n)).$$

The optimal expectation is obviously a non-decreasing function upper bounded by W, therefore it converges. Because of the above inequality, the optimal expectation has the same limit than the expectation of Algorithm 1, whose expectation is thus asymptotically optimal.

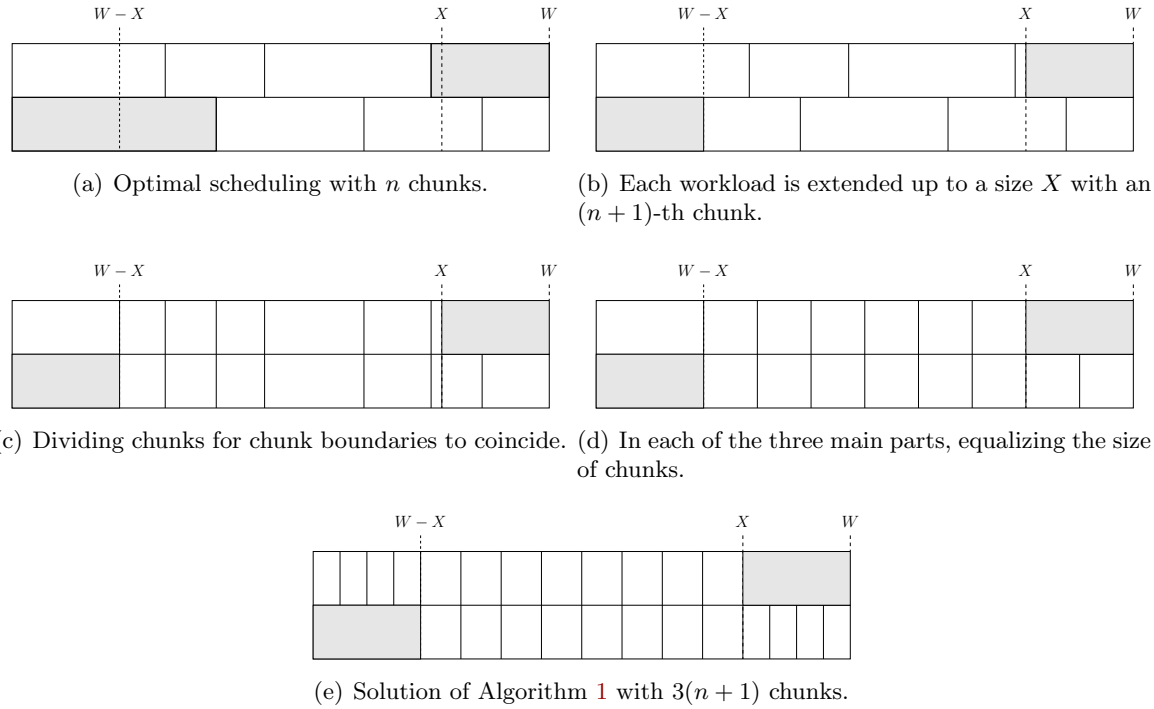
3. Case: $X < W < 2X$.

   As for the previous case, we must prove two things: that the proposed schedule is asymptotically optimal and that its expectation for jobdone is what we claim it is. Let us take any positive integer n. Then, the expectation of jobdone under Algorithm 1 is no greater than this expectation under the optimal scheduling:

   $$E^{(\mathrm{f},2)}(W,n) \geq E^{(\mathrm{f},2)}(W, \text{Algorithm 1}(n)).$$

   Following the series of transformations illustrated by Figure 5, we show that the optimal scheduling with n chunks is not a better expectation than the solution of Algorithm 1 with $3(n+1)$ chunks. Each transformation is a non-decreasing transformation from the point of view of the expectation of jobdone.

   We start from an optimal scheduling for n chunks satisfying Theorem 4 (see Figure 5(a)). By definition of X any computer-workload no smaller than X is obviously strictly suboptimal. In the first transformation (see Figure 5(b)) we add a $(n+1)$-th chunk to the workload of each computer for each computer-workload to be exactly equal to X. Obviously, this transformation does not change the expectation. Then, we subdivide the chunks so that the boundaries of the chunks of a computer coincide with the boundaries of the chunks of the other computer (see Figure 5(c)). For a formal description of this



(a) Optimal scheduling with n chunks.

(b) Each workload is extended up to a size X with an $(n+1)$-th chunk.

(c) Dividing chunks for chunk boundaries to coincide.

(d) In each of the three main parts, equalizing the size of chunks.

(e) Solution of Algorithm 1 with $3(n+1)$ chunks.

Figure 5: Series of schedule transformations to prove the asymptotic optimality of Algorithm 1 when $X < W < 2X$.

process, see the proof of the case $W \leq X$. Subdividing chunks does not decrease the expectation. We must still count how many chunks we may have in each of the three main parts of the workload, that is, in the intervals $[0, W-X]$, $[W-X, X]$, and $[X, W]$. Note that each of the interval bounds is a chunk boundary. The chunk boundaries in $[0, W-X]$ can only come from original chunks of P_1 and from the bound of the $(n+1)$-th chunk we added to P_2 (which gives a bound at $W - X$). Therefore, there are at most $n+1$ chunks in $[0, W-X]$. The same is true for $[X, W]$. Now, looking at the interval $[W-X, X]$, all the boundaries of the chunks $\mathcal{W}_{1,2}$, ..., $\mathcal{W}_{1,n}$ could have strictly lied in this interval. The same thing is true for the chunks $\mathcal{W}_{2,2}$, ..., $\mathcal{W}_{2,n}$. Therefore, in the worst case, there can be $2n$ chunk boundaries strictly between $W - X$ and X. This gives us at most $2n+1$ chunks in the interval $[W-X, X]$. Algorithm 1(3(n+1)) builds a solution with $n+1$ chunks in the interval $[0, W-X]$, $2n+2$ chunks in the interval $[W-X, X]$, and $n+1$ chunks in the interval $[X, W]$. Therefore, in none of the three main parts of the schedule does it have less chunks than the solution we just built.

The third transformation is done interval per interval. In each of the intervals $[0, W-X]$, $[W-X, X]$, and $[X, W]$ we distribute the interval workload equally among the chunks (see Figure 5(d)). From the proof of the case $W \leq X$ we know that this transformation does not decrease the expectation when it is solely applied to the interval $[W-X, X]$. Therefore, what only remains to prove is that this transformation when solely applied to the interval $[0, W-X]$ does not decrease the expectation (the fact that the different intervals do not impact each other is due to our failure model and to the fact that no chunk simultaneously strictly belong to two intervals). To establish the desired result

we only consider two consecutive chunks of P_1, $\mathcal{V}_{1,i}$ and $\mathcal{V}_{1,i+1}$ belonging in the interval $[0, W-X]$ (and thus which do not intersect the workload of P_2). The contribution of these two chunks to the expectation is:

$$\mathcal{E} = |\mathcal{V}_{1,i}| \left(1 - \sum_{j=1}^{i} |\mathcal{V}_{1,j}|\kappa\right) + |\mathcal{V}_{1,i+1}| \left(1 - \sum_{j=1}^{i+1} |\mathcal{V}_{1,j}|\kappa\right).$$

Using the notations $L = |\mathcal{V}_{1,i}| + |\mathcal{V}_{1,i+1}|$ and $V = \sum_{j=1}^{i-1} \mathcal{V}_{1,j}$, we have:

$$\begin{aligned}\mathcal{E} &= |\mathcal{V}_{1,i}|\left(1 - (V + |\mathcal{V}_{1,i}|)\kappa\right) + (L - |\mathcal{V}_{1,i}|)\left(1 - (V+L)\kappa\right) \\ &= -|\mathcal{V}_{1,i}|^2\kappa + L|\mathcal{V}_{1,i}|\kappa + L(1-(V+L)\kappa).\end{aligned}$$

This last expression is obviously maximized when $|\mathcal{V}_{1,i}| = \frac{L}{2}$, that is, when the two consecutive chunks have the same size.

The last transformation increases the number of same-size jobs in each of the three phases of the scheduling for these numbers to respectively be $n+1$, $2(n+1)$, and $n+1$ (see Figure 5(b)). We already know, from the study of the case $W \leq X$, that this is not decreasing the expectation for the chunks in the interval $[W-X, X]$. We now show that this is also the case for chunks in the interval $[0, W-X]$. The cumulative expectation for the m equal-size chunks of the interval $[0, W-X]$ is:

$$\begin{aligned}\mathcal{E} &= \textstyle\sum_{i=1}^{m} \frac{W-X}{m}\left(1 - \sum_{j=1}^{i} \frac{W-X}{m}\kappa\right) \\ &= (W-X) - \frac{(W-X)^2\kappa}{m^2}\sum_{i=1}^{m} i \\ &= (W-X) - \left(\tfrac{1}{2} + \tfrac{1}{2m}\right)(W-X)^2\kappa\end{aligned}$$

which is obviously increasing with m.

The expectation of jobdone, with $l = \left\lfloor \frac{n}{3} \right\rfloor$, for the scheduling of Algorithm 1(n) is then equal to:

$$\begin{aligned}\mathcal{E} = \;& \textstyle\sum_{i=1}^{l} \frac{W-X}{l}\left(1 - i\frac{W-X}{l}\kappa\right) \\ & + \sum_{i=1}^{2l} \frac{2X-W}{2l}\left(1 - \left(W - X + i\frac{2X-W}{2l}\right)\kappa\left(X - (i-1)\frac{2X-W}{2l}\right)\kappa\right) \\ & + \sum_{i=1}^{l} \frac{W-X}{l}\left(1 - i\frac{W-X}{l}\kappa\right) \\ =\; & 2W - \tfrac{1}{3}X - W^2\kappa + \frac{W^3\kappa^2}{6} \\ & + \tfrac{1}{l}\left(\left(1+\tfrac{1}{l}\right)W - \left(1+\tfrac{2}{3l}\right)X - \tfrac{1}{2l}W^2\kappa - \tfrac{1}{4}\left(1-\tfrac{1}{3l}\right)W^3\kappa^2\right).\end{aligned}$$

□

5 Scheduling for p Remote Computers

We finally turn to the general case, wherein there are p remote computers. We have discovered this case of general p to be much more difficult than the already challenging case $p = 2$, so we devote our efforts here to searching for *efficient heuristic schedules*.

In order to appreciate how hard it is to extend the case $p = 2$ even to $p = 3$, the reader is invited to seek an analogue of Theorem 4 for $p = 3$. As one example, we have not discovered a 3-computer analogue of "mirroring," and our attempts to do so have all fallen to unobvious schedules such as those discussed after Theorem 5.

Because of the difficulty of the general scheduling problem, we adopt a pragmatic approach, by focusing only on the linear risk model, and by restricting attention to "well-structured" schedules that employ same size chunks.

> Our restriction to same-size chunks 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. This suggests that such chunking may be computationally beneficial. (2) This restriction greatly simplifies the specification and implementation of schedules for the case of general p, by imposing simplifying structure on this extremely hard 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 computer 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 that are tailored to the linear risk function—but we suggest how they can be adapted to other risk functions. We partition our scheduling problem into three subproblems, based on the size of the workload we wish to schedule. This partition—which acknowledges the futility of deploying a workslice of size $> X$ on any computer, in the light of our interruption model—gives us two easy subproblems and one challenging one that will occupy the rest of our attention.

W **is "very small."** When $W \leq X$, we deploy the entire workload in a single slice, which we replicate 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. (We assume here that work is not prioritized, so we do not care *which* pX units we deploy.)

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 $Z = \min(W, pX)$ units of work in to 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.

> Among the ways in which we have tailored the preceding scenario to the linear risk function is by demanding that slices have size $\leq X$. For general risk functions, we would introduce a parameter λ that specifies the maximum probability of interruption that the user would allow for a slice. We would then use λ to compute the maximum allowable slice size maxsl by insisting that $Pr(\mathsf{maxsl}) = \lambda$. For

instance if $\lambda = 1/2$, then with the linear risk function we would set $\mathsf{maxsl} = \frac{1}{2}X$, while with the exponential risk function we would set $\mathsf{maxsl} = (\ln 2)X$. The amount of work we actually deploy would now be $Z = \min(W,\ p \times \mathsf{maxsl})$. This would mandate using $q = \lceil Z/\mathsf{maxsl} \rceil$ slices, of common size $\mathsf{sl} = Z/q$.

We now specify the partition procedure, Algorithm 2, which takes three inputs: the total amount of work W, the number p of computers, and the maximum allowable risk λ. The algorithm returns the number of slices, their sizes, and the number of remote computers that each slice is deployed to.

Algorithm 2: The partitioning algorithm for p computers.

```
1  procedure Partition(W, p, λ)
2  begin
3  |  /*Determine maxsl such that Pr(maxsl) = λ*/
4  |  Z ← min(W, p × maxsl)
5  |  q ← ⌈Z/maxsl⌉
6  |  sl ← Z/q
7  |  r ← p mod q; s ← q − r
8  |  Partition the computers into r coteries of cardinality ⌊p/q⌋ + 1 each and
9  |     s coteries of cardinality ⌊p/q⌋ each
10 end
```

5.2 The Orchestration Phase

The partition phase has left us with independent slices of work, each of size $\mathsf{sl} \leq \mathsf{maxsl}$, that will be executed by disjoint coteries of computers. All slices will be partitioned into n chunks of common size $\omega = \mathsf{sl}/\mathsf{n}$, where the "checkpointing granularity" n is specified by the user. For each coterie Γ of computers, each chunk assigned to coterie Γ will be executed by all $\mathsf{g}_\Gamma \in \{\lfloor p/q \rfloor,\ 1 + \lfloor p/q \rfloor\}$ computers in the coterie. Our challenge is to determine how to orchestrate the g_Γ executions of each chunk—i.e., to determine when (at which times 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. The remainder of our study is dedicated to this orchestration phase.

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 of computers operates independently of all others, we can specify the overall schedule coterie by coterie. For each coterie Γ and its associated slice, we represent a possible schedule for Γ's executing the slice via a table such as Table 1; we call these tables *execution charts*. Rows in these charts enumerate the computers in the associated coterie Γ, and columns enumerate the indices of the chunks into which coterie Γ's slice is chopped. Chart-entry $C_{i,j}$ is the step at which chunk j is processed by computer P_i.

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: An execution chart for a coterie of four computers. In this example, chunk 5 is executed by P_2 at step $C_{2,5} = 11$.

Any $\mathsf{g} \times \mathsf{n}$ integer matrix whose rows are permutations of $[1..\mathsf{n}]$ can be used as the execution chart for a valid schedule for the slice, under which each P_i executes each chunk j (specifically, 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 a 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

$$\begin{aligned} E(\mathsf{sl},\mathsf{n}) &= (\mathsf{sl}/\mathsf{n}) \sum_{j=1}^{\mathsf{n}} \left(1 - \prod_{i=1}^{\mathsf{g}} Pr\left(C_{i,j}\mathsf{sl}/\mathsf{n}\right)\right) \\ &= \mathsf{sl}\left(1 - \frac{1}{\mathsf{n}}\left(\frac{\mathsf{sl}\kappa}{\mathsf{n}}\right)^{\mathsf{g}} \sum_{j=1}^{\mathsf{n}} \prod_{i=1}^{\mathsf{g}} C_{i,j}\right). \end{aligned} \tag{13}$$

The last expression, (13), is specific to the linear-risk model: because each $C_{i,j} \leq n$, we have $Pr(C_{i,j}\omega) = C_{i,j}\omega\kappa$ under the linear-risk model, so we need not take the minimum of the last expression with 1 (as in (1)).

We derive the following upper bound:

Proposition 1.

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

Proof. Let $\mathsf{cp}_j = \prod_{i=1}^{\mathsf{g}} C_{i,j}$ be the j-th column product in the chart. From the expression of $E(\mathsf{sl},\mathsf{n})$, we see that it is maximum when the sum of the n column products is minimum. But the product of the column products is constant, because each row is a permutation of $[1..\mathsf{n}]$: we have $\prod_{j=1}^{\mathsf{n}} \mathsf{cp}_j = (\mathsf{n}!)^{\mathsf{g}}$. The sum is minimum when all products are equal (to $(\mathsf{n}!)^{\frac{\mathsf{g}}{\mathsf{n}}}$), whence the result. □

Stirling's formula gives a useful approximation of the upper bound when n is large:

$$E_{\max} \approx \mathsf{sl} \cdot \left(1 - \left(\frac{\mathsf{sl}.\kappa}{e}\right)^{\mathsf{g}}\right).$$

5.2.2 Group schedules: introduction

Referring back to Table 1, we observe that chunks 1, 2, 3, and 4 are always executed at the same steps, by different computers; the same is true for chunks $5, 6, 7, 8$ as a group, and for chunks $9, 10, 11, 12$ as a group. The twelve chunks of the slice thus partition naturally into three *groups*. By respecifying the schedule of Table 1 as the *group(-oriented)* schedule

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

Table 2: Execution chart for a group-oriented schedule. Rows represent time steps for the first computer in each group associated with each column; the remaining computers' schedules are obtained by cyclic downward permutations of the rows.

of Table 2, we significantly simplify the specification. Note that the meanings of rows and columns have changed in this re-orientation: compare Tables 1 and 2 as we describe the changes. In the group(-oriented) execution chart of Table 2, each column corresponds to a group of chunks; entry (i, j) of the chart specifies the step at which each computer executes its ith chunk within group j. The schedule for computer P_j, where $j \in \{2, 3, 4\}$, is obtained by cyclically permuting (downward) the schedule for P_1 $j - 1$ times. The important feature here is that this orchestration has each computer attempt to execute each chunk 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, as 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, each 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. It is tacitly assumed that chunk-indices within each group are cyclically permuted (downward) at each step, so that each chunk ends up being processed by each computer. Thus, in order 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:

$$E(\mathsf{sl}, \mathsf{n}) \;=\; \mathsf{sl}\left(1 - \frac{\mathsf{g}}{\mathsf{n}}\left(\frac{\mathsf{sl}\cdot\kappa}{\mathsf{n}}\right)^{\mathsf{g}} \sum_{j=1}^{\mathsf{n}/\mathsf{g}} \prod_{i=1}^{\mathsf{g}} \widehat{C}_{i,j}\right). \tag{14}$$

The preceding expression exposes the importance of the constant

$$\mathsf{K}^{(\Theta)} \;=\; \sum_{j=1}^{\mathsf{n}/\mathsf{g}} \prod_{i=1}^{\mathsf{g}} \widehat{C}_{i,j}^{(\Theta)}$$

as a measure of a group schedule Θ's performance; to wit,

$$E^{(\Theta)}(\mathsf{sl}, \mathsf{n}) \;=\; \mathsf{sl} - \mathsf{K}^{(\Theta)} \cdot \frac{\mathsf{g}}{\kappa}\left(\frac{\mathsf{sl}\kappa}{\mathsf{n}}\right)^{\mathsf{g}+1}. \tag{15}$$

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

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—which is the best we have been able to prove—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*,*

$$E^{(\Theta)}(\mathsf{sl},\mathsf{n}) \;\leq\; E_{\max} \;=\; \mathsf{sl} - \frac{\mathsf{sl}^{\mathsf{g}+1}\kappa^{\mathsf{g}}(\mathsf{n}!)^{\mathsf{g}/\mathsf{n}}}{\mathsf{n}^{\mathsf{g}}}.$$

Proof. Let $\mathsf{cp}_j = \prod_{i=1}^{\mathsf{g}} G_{i,j}$ be the j-th column product. As before, $E(\mathsf{sl},\mathsf{n})$ is maximum when the sum of the $\frac{\mathsf{n}}{\mathsf{g}}$ column products is minimum. The product of these column products is equal to a constant $(\mathsf{n}!)$. The sum is minimum when all products are equal to $(\mathsf{n}!)^{\frac{\mathsf{g}}{\mathsf{n}}}$, hence the same result as for Proposition 1. □

Note that Proposition 2 affords us an easy lower bound, $\mathsf{K}_{\min}$ on the K value of any group schedule with the parameters g and n:

$$\mathsf{K}_{\min} \;=\; \left\lceil \frac{\mathsf{n}}{\mathsf{g}}(\mathsf{n}!)^{\mathsf{g}/\mathsf{n}} \right\rceil .$$

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, in the case $p = 2$, for *every* risk function, it is advantageous to have the remote computers process the work they share "in opposite orders" (Theorem 4). We now specify and compare the performance of five group schedules whose chunk-scheduling regimens seem to be a good match for the way the linear risk function "predicts" interruptions. We specify each schedule Θ via its group execution chart $\widehat{C}^{(\Theta)}$—see Fig. 6—and we represent the performance of each schedule Θ via its performance constant $\mathsf{K}^{(\Theta)}$. The beneficent structures of these schedules is evidenced by our ability to present explicit symbolic expressions for their K constants.

Cyclic scheduling (Fig. 6(a)). Under this simplest scheduling regimen, Θ_{cyclic}, groups are executed sequentially, in a round-robin fashion. Specifically, the chunks of group j are executed at steps j, $j + \mathsf{n}/\mathsf{g}$, $j + 2\mathsf{n}/\mathsf{g}$, and so on. We find that

$$\mathsf{K}^{(\Theta_{\mathsf{cyclic}})} \;=\; \sum_{j=1}^{\mathsf{n}/\mathsf{g}} \prod_{k=0}^{\mathsf{g}-1} (j + k\mathsf{n}/\mathsf{g}) \,.$$

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

(a) Cyclic: $\mathsf{K} = 3104$

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

(b) Reverse: $\mathsf{K} = 2368$

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

(c) Mirror: $\mathsf{K} = 2572$

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

(d) Snake: $\mathsf{K} = 2464$

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

(e) Fat snake: $\mathsf{K} = 2364$

Figure 6: Five group schedules with their associated K values. For this instance, $\mathsf{K}_{\min} = 2348$.

The weakness of Θ_{cyclic} is that chunks in low-index groups have a higher probability of being completed successfully than do chunks in high-index groups—because chunks remain in the same relative order throughout the computation. The remaining schedules that we consider aim to compensate for this imbalance via different intuitively motivated strategies.

Reverse scheduling (Fig. 6(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. The schedule 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.) Under $\Theta_{\mathsf{reverse}}$, 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. We find that

$$\mathsf{K}^{(\Theta_{\mathsf{reverse}})} = \sum_{j=1}^{\mathsf{n}/\mathsf{g}} j \times \prod_{k=1}^{\mathsf{g}-1} \left((k+1)\mathsf{n}/\mathsf{g} - j + 1\right).$$

Mirror scheduling (Fig. 6(c)). The mirror schedule Θ_{mirror}, which is defined only when g is even, represents a compromise between the cyclic and reverse scheduling strategies. Θ_{mirror} compensates for the imbalance in likelihood of completion only during the second half of the computation. Specifically, Θ_{mirror} mimics Θ_{cyclic} for the first $\mathsf{g}/2$ phases of processing a group,

and it mimics $\Theta_{\mathsf{reverse}}$ for the second $\mathsf{g}/2$ phases. We find that

$$\mathsf{K}^{(\Theta_{\mathsf{mirror}})} = \sum_{j=1}^{\mathsf{n/g}} \prod_{k=0}^{\frac{1}{2}\mathsf{g}-1} (j + k\mathsf{n/g})\,((p-k)\mathsf{n/g} - j + 1)\,.$$

Snake-like scheduling (Fig. 6(d)). Our fourth schedule, Θ_{snake}, compensates for the imbalance of the cyclic schedule by mimicking Θ_{cyclic} at every odd-numbered step and mimicking $\Theta_{\mathsf{reverse}}$ at every even-numbered step, thereby lending a snake-like structure to the execution chart $\widehat{C}^{(\Theta_{\mathsf{snake}})}$. We find that

$$\mathsf{K}^{(\Theta_{\mathsf{snake}})} = \sum_{j=1}^{\mathsf{n/g}} \prod_{k=0}^{\frac{1}{2}\mathsf{g}-1} (j + 2k\mathsf{n/g})\,(2(k+1)\mathsf{n/g} - j + 1)\,.$$

Fat snake-like scheduling (Fig. 6(e)). Our final, fifth schedule, $\Theta_{\mathsf{fat-snake}}$, qualitatively adopts the same strategy as does Θ_{snake}, but it slows down the return phase of the latter schedule. Consider, for illustration, 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 of Fat snake 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}. We find that

$$\mathsf{K}^{(\Theta_{\mathsf{fat-snake}})} = \sum_{j=0}^{\mathsf{n/g}-1} \prod_{k=0}^{\frac{1}{3}\mathsf{g}-1} (1 + j + 3k\mathsf{n/g})\,(3(k+1)\mathsf{n/g} - 2j - 1)\,(3(k+1)\mathsf{n/g} - 2j)\,.$$

We derive the following performance bounds for these five schedules

Proposition 3. *The values of* $\mathsf{K}^{(\Theta)}$ *for our five scheduling algorithms satisfy the following lower and upper bounds:*

$$\begin{aligned}
\frac{1}{\mathsf{n}} &\le \frac{\mathsf{K}^{(\Theta_{\mathsf{cyclic}})}}{\mathsf{g!}\,(\mathsf{n/g})^{\mathsf{g}+1}} \le 1 \\
\frac{1}{2\mathsf{g}} &\le \frac{\mathsf{K}^{(\Theta_{\mathsf{reverse}})}}{\mathsf{g!}\,(\mathsf{n/g})^{\mathsf{g}+1}} \le \tfrac{1}{2}(\mathsf{n}+\mathsf{g}) \\
\frac{1}{\mathsf{n}} &\le \frac{\mathsf{K}^{(\Theta_{\mathsf{mirror}})}}{\mathsf{g!}\,(\mathsf{n/g})^{\mathsf{g}+1}} \le 1 \\
\frac{\mathsf{g}}{\mathsf{n}^2} &\le \frac{\mathsf{K}^{(\Theta_{\mathsf{snake}})}}{\mathsf{g!}\,(\mathsf{n/g})^{\mathsf{g}+1}} \le 1 \\
\frac{1}{(\mathsf{g}-1)\mathsf{n}} &\le \frac{\mathsf{K}^{(\Theta_{\mathsf{fat-snake}})}}{\mathsf{g!}\,(\mathsf{n/g})^{\mathsf{g}+1}} \le \mathsf{g}
\end{aligned}$$

Proof. The calculations are straightforward. For the Cyclic schedule, we have

$$\mathsf{K}_{\mathsf{cyclic}} = \sum_{j=1}^{\frac{\mathsf{n}}{\mathsf{g}}} \prod_{k=0}^{\mathsf{g}-1} \left(j + k\frac{\mathsf{n}}{\mathsf{g}} \right).$$

We derive the lower bound by replacing index j by 0 in the summation (except in the term $k = 0$ where we replace j by 1):

$$\mathsf{K}_{\mathsf{cyclic}} \geq \frac{\mathsf{n}}{\mathsf{g}} \left(\prod_{k=1}^{\mathsf{g}-1} k\frac{\mathsf{n}}{\mathsf{g}} \right) = (\mathsf{g}-1)! \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}} = \frac{1}{\mathsf{n}}\mathsf{g}! \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1}.$$

Similarly, we let $j = \frac{\mathsf{n}}{\mathsf{g}}$ in each term of the summation to get the upper bound. We proceed in a similar way for the other three variants.

We explicit the computations for Fat snake, as they are a bit less obvious. For the lower bound we have:

$$\begin{aligned}
\mathsf{K}_{\mathsf{fat-snake}} &= \textstyle\sum_{j=0}^{\frac{\mathsf{n}}{\mathsf{g}}-1} \prod_{k=0}^{\frac{\mathsf{g}}{3}-1} \left(1 + j + 3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}} - 2j - 1\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}} - 2j\right) \\
&\geq \textstyle\sum_{j=0}^{\frac{\mathsf{n}}{\mathsf{g}}-1} \prod_{k=0}^{\frac{\mathsf{g}}{3}-1} \left(1 + 3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}} - 2\frac{\mathsf{n}}{\mathsf{g}} + 1\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}} - 2\frac{\mathsf{n}}{\mathsf{g}} + 2\right) \\
&\geq \frac{\mathsf{n}}{\mathsf{g}} \textstyle\prod_{k=0}^{\frac{\mathsf{g}}{3}-1} \left(1 + 3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \\
&\geq \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^3 \textstyle\prod_{k=1}^{\frac{\mathsf{g}}{3}-1} \left(3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \\
&\geq \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^3 \textstyle\prod_{k=1}^{\frac{\mathsf{g}}{3}-1} \left((3k-1)\frac{\mathsf{n}}{\mathsf{g}}\right) \left(3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \\
&= \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}} (\mathsf{g}-2)!
\end{aligned}$$

For the upper bound we derive:

$$\begin{aligned}
\mathsf{K}_{\mathsf{fat-snake}} &= \textstyle\sum_{j=0}^{\frac{\mathsf{n}}{\mathsf{g}}-1} \prod_{k=0}^{\frac{\mathsf{g}}{3}-1} \left(1 + j + 3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}} - 2j - 1\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}} - 2j\right) \\
&\leq \textstyle\sum_{j=0}^{\frac{\mathsf{n}}{\mathsf{g}}-1} \prod_{k=0}^{\frac{\mathsf{g}}{3}-1} \left(\frac{\mathsf{n}}{\mathsf{g}} + 3k\frac{\mathsf{n}}{\mathsf{g}}\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \left(3(k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \\
&\leq \frac{\mathsf{n}}{\mathsf{g}} \textstyle\prod_{k=0}^{\frac{\mathsf{g}}{3}-1} \left((3k+1)\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+3)\frac{\mathsf{n}}{\mathsf{g}}\right) \left((3k+3)\frac{\mathsf{n}}{\mathsf{g}}\right) \\
&\leq \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1} \left(\textstyle\prod_{k=0}^{\frac{\mathsf{g}}{3}-2} (3k+2)(3k+3)(3k+4)\right) (\mathsf{g}-2)\mathsf{g}^2 \\
&= \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1} (\mathsf{g}-2)!(\mathsf{g}-2)\mathsf{g}^2 \leq \left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1} (\mathsf{g})!\mathsf{g}
\end{aligned}$$

□

From the size of its bounds on $\mathsf{K}^{(\Theta_{\mathsf{snake}})}$, Proposition 3 suggests that schedule Θ_{snake} may be the most efficient of the five group-scheduling algorithms we have considered, especially when we checkpoint often, i.e., when n is large. We evaluate this possibility via the experiments reported at the end of this subsection. While still focusing on mathematical analyses of our schedules, though, we use Stirling's formula to derive more evocative bounds on $\mathsf{K}^{(\Theta_{\mathsf{snake}})}$: $\mathsf{K}_{\min} \leq \mathsf{K}^{(\Theta_{\mathsf{snake}})} \leq \mathsf{K}_{\mathsf{upper}}$, where

$$\mathsf{K}_{\min} \approx \frac{e}{g}\left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1} \quad \text{and} \quad \mathsf{K}_{\mathsf{upper}} = \mathsf{g}!\left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1} \approx \frac{e\sqrt{2\pi}}{\sqrt{g}}\left(\frac{\mathsf{n}}{\mathsf{g}}\right)^{\mathsf{g}+1}.$$

We conclude this subsection by adding a last element to our set of group schedules. The resulting "greedy" procedure strives to iteratively balance the probability of success for each group of chunks. As we do not get any asymptotic estimation for Greedy, we content ourselves with a numerical estimate.

Greedy scheduling (Table 3). 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 the product $\Pi_{i=1}^{k} \widehat{C}_{ij}^{(\mathsf{greedy})}$ of the entries in column j of the chart. The idea is to sort current column products and to assign the smallest time-step to the largest product, and so on. Table 3 illustrates a computation with $\mathsf{n} = 12$ and $\mathsf{g} = 4$. In this example, Θ_{greedy} is identical to $\Theta_{\mathsf{reverse}}$, hence achieves the same performance constant $\mathsf{K}^{(\Theta_{\mathsf{greedy}})} = \mathsf{K}^{(\Theta_{\mathsf{reverse}})} = 2368$.

Step 1	1	2	3
CCP	1	2	3
Step 2	6	5	4
CCP	6	10	12
Step 3	9	8	7
CCP	54	80	84
Step 4	12	11	10
CCP	6	880	12

Table 3: A computation by Θ_{greedy}. CCP denotes the *Current Column Product*.

For the record, and for the curious reader: Table 4 provides an example for which none of our group schedules is optimal, and Table 5 shows an example for which Θ_{greedy} differs from, and outperforms, $\Theta_{\mathsf{reverse}}$.

1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
4	5	6	6	5	4	6	5	4	8	6	4	8	5	4
7	8	9	7	8	9	9	8	7	9	7	5	9	7	6
$\mathsf{K}^{(\Theta_{\mathsf{cyclic}})} = 270$			$\mathsf{K}^{(\Theta_{\mathsf{snake}})} = 230$			$\mathsf{K}^{(\Theta_{\mathsf{reverse}})} = \mathsf{K}^{(\Theta_{\mathsf{greedy}})} = 218$			$\mathsf{K}^{(\Theta_{\mathsf{fatsnake}})} = 216$			$\mathsf{K}_{\mathsf{optimal}} = \mathsf{K}_{\min} = 214$		

Table 4: Comparing group schedules for $\mathsf{n} = 9$ and $\mathsf{g} = 3$. (Θ_{mirror} is missing because g is odd). Here $\Theta_{\mathsf{reverse}}$ and Θ_{greedy} are identical. The optimal schedule achieves the bound $\mathsf{K}_{\min}$.

Numerical evaluation We ran all six of our scheduling heuristics on all problems where $g \in [2, 100]$, $n \in [2 * g, 1000]$, and g divides n; altogether, this corresponds to 4032 instances. We report in Table 6 two series of statistics. In the *Relative* series, we form the ratio of the K value of a given heuristic on a given instance over the lowest K value found for that instance among all the tested heuristics. For the *Absolute* series, we form the ratio with $\mathsf{K}_{\min}$. In Table 6 we also report the *best-of* heuristic that, on each instance, runs the six other algorithms and picks the best answer.

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

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

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

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

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

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

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

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

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

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

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 5: Comparing group schedules for $\mathsf{n} = 20$ and $\mathsf{g} = 4$. Here the most efficient schedules are $\Theta_{Fatsnake}$, Θ_{greedy}, and $\Theta_{\mathsf{reverse}}$ (in this order). The lower bound is $\mathsf{K}_{\min} = 23780$.

	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%
Worm	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 6: Statistics on the K value of all heuristics for $2 \leq g \leq 100$ and $2g \leq n \leq 1000$ (minimum, maximum, average value and standard deviation over the 4032 instances).

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, its performance constant 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

To this point, we have assumed that the number n of chunks per computer was given to us. In fact, we show now that (happily) one does not have to guess at this value. 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 under the charged-initiation model, from that schedule's analogous expectation under the free-initiation model.

Theorem 7. (p remote computers: charged-initiation model)
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 $\mathsf{sl}^{(c)} \leq \min\{\mathsf{sl}, X-\mathsf{n}\varepsilon\}$, *then:*

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

Proof. Because of the condition $\mathsf{sl}^{(c)} \leq \min\{\mathsf{sl}+\mathsf{n}\varepsilon, X\}$, Eq. (15) can be used with $\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon$ instead of sl. After rewritting we get:

$$\frac{\mathsf{n}}{\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon}\left(E^{(f,\mathsf{n})}(\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon,\mathcal{C})-\mathsf{sl}^{(c)}-\mathsf{n}\varepsilon\right) = -\mathsf{K}^{(\mathcal{C})}\left(\left(\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}+\varepsilon\right)\kappa\right)^{\mathsf{g}}.$$

Now, we can directly compute $E^{(c,\mathsf{n})}(\mathsf{sl}^{(c)},\mathcal{C})$:

$$\begin{aligned}
E^{(c,\mathsf{n})}(\mathsf{sl}^{(c)},\mathcal{C}) &= \frac{\mathsf{sl}^{(c)}}{\mathsf{n}}\sum_{j=1}^{\mathsf{n}}\left(1-\prod_{i=1}^{\mathsf{g}}Pr^{(c)}\left(C_{i,j}\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}\right)\right)\\
&= \frac{\mathsf{sl}^{(c)}}{\mathsf{n}}\sum_{j=1}^{\mathsf{n}}\left(1-\prod_{i=1}^{\mathsf{g}}C_{i,j}\left(\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}+\varepsilon\right)\kappa\right)\\
&= \mathsf{sl}^{(c)}-\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}\left(\left(\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}+\varepsilon\right)\kappa\right)^{\mathsf{g}}\sum_{j=1}^{\mathsf{n}}\prod_{i=1}^{\mathsf{g}}C_{i,j}\\
&= \mathsf{sl}^{(c)}-\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}\left(\left(\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}+\varepsilon\right)\kappa\right)^{\mathsf{g}}\mathsf{K}^{(\mathcal{C})}\\
&= \mathsf{sl}^{(c)}+\frac{\mathsf{sl}^{(c)}}{\mathsf{n}}\frac{\mathsf{n}}{\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon}\left(E^{(f,\mathsf{n})}(\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon,\mathcal{C})-\mathsf{sl}^{(c)}-\mathsf{n}\varepsilon\right)\\
&= \frac{\mathsf{sl}^{(c)}}{\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon}E^{(f,\mathsf{n})}(\mathsf{sl}^{(c)}+\mathsf{n}\varepsilon,\mathcal{C}).
\end{aligned}$$

□

Now we can 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 non-decreasing with n under the free-initiation model.) We can, then, 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, in which case m is smaller than the optimum n. The binary search can be safely performed in the interval $[1..X/\varepsilon]$.

6 Experiments

We have performed a suite of simulation experiments in order to gain insight into the performance of the group heuristics on large simulated platforms that are subject to unrecoverable

interruptions. We report only on the observed behavior of Θ_{greedy} for two reasons, first because of its preeminence in the experiment reported in Table 6, and second, because our simulations show only small differences among our six heuristics. The source code for all six group heuristics can be found at `http://graal.ens-lyon.fr/~abenoit/code/failure.c`.

6.1 The Experimental Plan

We use randomly generated platforms made of p computers. In all experiments, we set $\kappa = 1$, and we choose the times for interruptions randomly chosen between 0 and 1, following a uniform distribution. The size of the workload, W_{tot}, varies between 1 and p. $W_{tot} = 1$ represents the case in which all computers can potentially do all the work before being interrupted; $W_{tot} = p$ represents the case in which we can do no better than deploy one slice of size 1 on each computer (which will then compute until it is interrupted).

The key parameters in our experiment are: the number of computers, p; the total amount of work W_{tot}; the chunk size, cs; and the start-up cost ε. In the experiments, three of these parameters are fixed while the fourth one varies.

We compare several heuristics:

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 the number of chunks is a multiple of p, this heuristic is identical to H2-norep, since the chunks assigned to a computer during the replication phase were already assigned to it previously.

H4-randomrep– This *random replication heuristic* distributes a total 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 5.2.2. Since our number of chunks n may not be a multiple of g, the last group of computers may not have a full g chunks to process. In this case, we ignore this last group once its computers have been assigned as many time-steps as its number of chunks.

H6-omniscient– This last *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.

6.2 Experimental Results

Heuristic	min	max	average	standard deviation
H1-brute	0.017199	1.000000	0.300934	0.316694
H2-norep	0.000000	1.000000	0.941108	0.156558
H3-cyclicrep	0.333333	1.000000	0.973718	0.070451
H4-randomrep	0.166667	1.000000	0.929911	0.106433
H5-groupgreedy	0.333333	1.000000	0.980068	0.055115
H6-omniscient	1.000000	1.000000	1.000000	0.000000

Table 7: Relative statistics on the overall simulations (2,696,800 instances): the work achieved by one heuristic on one instance is divided by the work achieved by the best heuristic for that instance.

In the following, we output the ratio W_{done}/W_{tot}, where W_{done} is the amount of work successfully completed by a heuristic, and W_{tot} is the total amount of work that had to be processed. The experiments have been conducted by averaging the result obtained on 100 different random configurations of the system (processor failure times).

(E1) Fixed p, cs and ε

In this first experiment, we analyse the impact of the workload on the heuristics. The total amount of work W_{tot} varies between 1 and the number of processors p, which are the two extreme cases. The chunk size is fixed at $1/n + \varepsilon$, where n is the number of chunks per unit of work. We fix the values $n = 97, 100, 997$ and $\varepsilon = 0.001, 0.00001$. Finally, we experiment with $p = 10$ and $p = 80$ processors.

Figure 7 presents the results with $\varepsilon = 0.001$ for all other combinations of parameters, while Figure 8 shows the behaviour when $\varepsilon = 0.00001$.

First we can notice that in all plots, as expected, H6-omniscient always return the best result, and H1-brute the worse one. 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 the number of processors, as can be observed on Figures 7(c), 7(d), 8(c) and 8(d). Notice that the total number of chunks increases with the total workload, which explains the particular behaviour of H3-cyclicrep in Figures 7(d) and 8(d).

Compared to these last two heuristics, H4-randomrep is generally getting better results than H2-norep for small workloads. This can be easily explained by the fact that replication allows to process chunks that fail in the case with no replication. However, when the workload increases, then H4 may distribute several times the same chunk while it succeeds on the first processor on which it was assigned. Thus, duplicated work will be completed. Of course, when the cyclic replication is useless, H4 is also better than H3 for small workloads.

Finally, the group greedy heuristic is in most cases the best one (aside from the optimal H6), at least when the workload is not too large. When the workload increases, since the number of chunks is not necessarily a multiple of the cardinality of groups $\mathtt{g}$, the handling of some chunks may not be optimized. Moreover, for a high workload, a cyclic distribution of the work will achieve a good result since not much redundant work can be done.

Even though this experiment did not aim to show the influence of the number of chunks and start-up cost, we can see on Figure 7 that the increase of the number of chunks leads to poorer performance: with 997 chunks the start-up cost almost is equal to the size of the chunk! Thus the performance of H1 drops under 0.5 even with a single processor. This is not true anymore when the start-up cost becomes negligible, as can be seen in Figure 8.

(E2) Fixed W_{tot}, cs and ε

In this second experiment, we study the behaviour of the heuristics when the number of processors increases up to 100. To compare on a fair basis, the total amount of work W_{tot} also increases with the number of processors, but it is always the same ratio: we let $W_{tot} = \alpha.p$, where α is fixed (hence the notation "fixed W_{tot}"). We consider the values $\alpha = 0.3$ and $\alpha = 0.7$, corresponding respectively to a low load and a high load of the system. The chunk size is still fixed to $1/n + \varepsilon$, where n is the number of chunks per unit of work. We fix the values $n = 50, 250$ and $\varepsilon = 0.001, 0.00001$.

Figure 9 shows these results, and it demonstrates that all heuristics scale well to large platforms, since the relative performance remains constant with the increase of the number of processors. Only heuristic H1-brute has a decreasing performance, because it does not exploit the fact that several processors can process different chunks in parallel. Also, we still observe the impact of the ratio number of chunks vs number of processors for the H3-cyclicrep heuristic, which often behaves as H2-norep in these settings, but sometimes manages to become close to H5-groupgreedy (see Figures 9(a), 9(c) and 9(e)). When the workload is more important, H2, H3 and H5 are close to each other, and their result is better than H1 and H4, and worse than the optimal H6 (see Figures 9(b), 9(d) and 9(f)). Notice that with a larger start-up cost as in Figure 9(f), these heuristics are very close to the optimal.

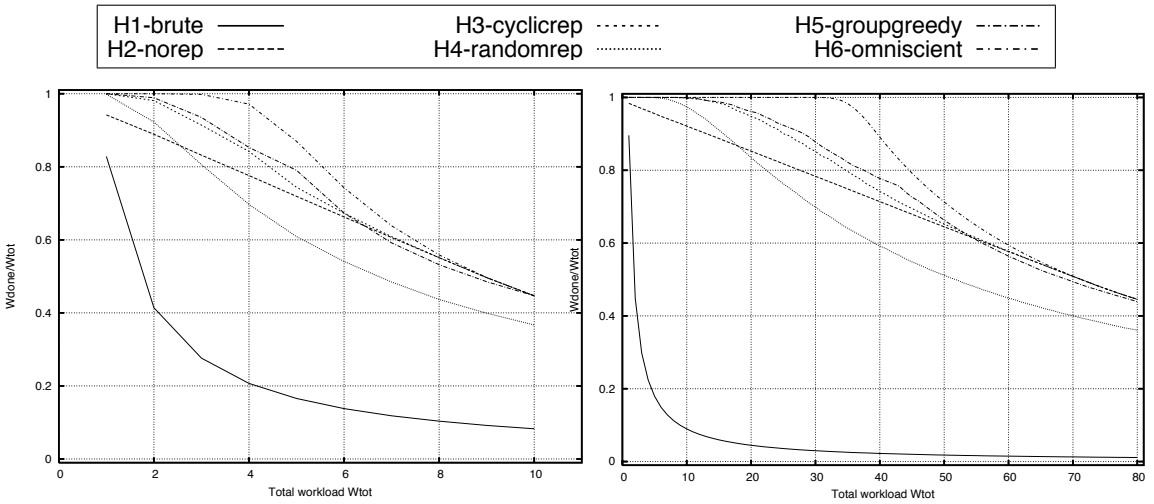
The conclusion of this experiment is that for various parameter settings, the heuristics scale very well to large platforms, which is an important result.

(E3) Fixed W_{tot}, p and ε

In this third experiment, both the total workload W_{tot} and the number of processors p are fixed. We consider two different settings. In the first one, $p = 10$ and $W_{tot} = 3, 7$ (see Figure 10): with a small number of processors we consider a small and large workload, without going into the extreme cases. In the second setting, $p = 80$ and $W_{tot} = 10, 70$ (see Figure 11): the goal is to see whether the number of processors has an impact of the number of chunks that should be chosen.

For each of these settings, several start-up costs are considered: $\varepsilon = 0.01$ (big start-up cost), $\varepsilon = 0.001$ (medium start-up cost), and $\varepsilon = 0.00001$ (negligible start-up cost). The goal of this experiment is to identify how many chunks should be used in order to maximize the amount of work done, depending on the setting.

When the start-up cost is negligible (see Figures 10(a), 10(b), 11(a), 11(b)), one should use a large number of chunks, since having small chunks reduces the loss that occurs when a processor fails. However, when the start-up cost increases, one should be more cautious because this cost decreases the performance of the solution. In the case with 10 processors, only a small number of chunks should be used when the start-up cannot be neglected. For big start-up costs (Figures 10(e) and 10(f)), the decrease of performance is very important.



(a) 10 processors, 97 chunks.

(b) 80 processors, 97 chunks.

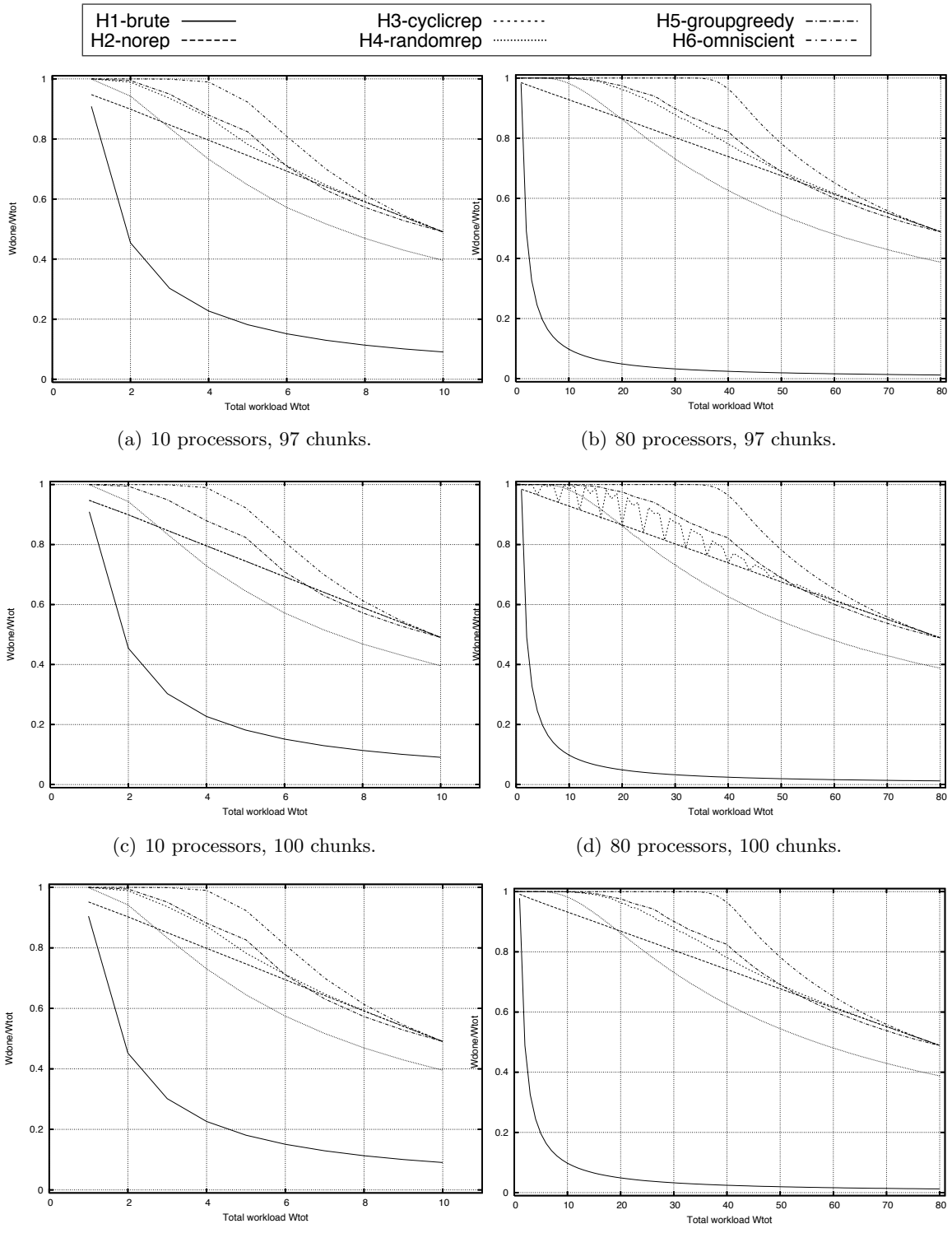
(c) 10 processors, 100 chunks.

(d) 80 processors, 100 chunks.

(e) 10 processors, 997 chunks.

(f) 80 processors, 997 chunks.

Figure 7: (E1) with $\varepsilon = 0.001$, varying W_{tot}.



(a) 10 processors, 97 chunks.

(b) 80 processors, 97 chunks.

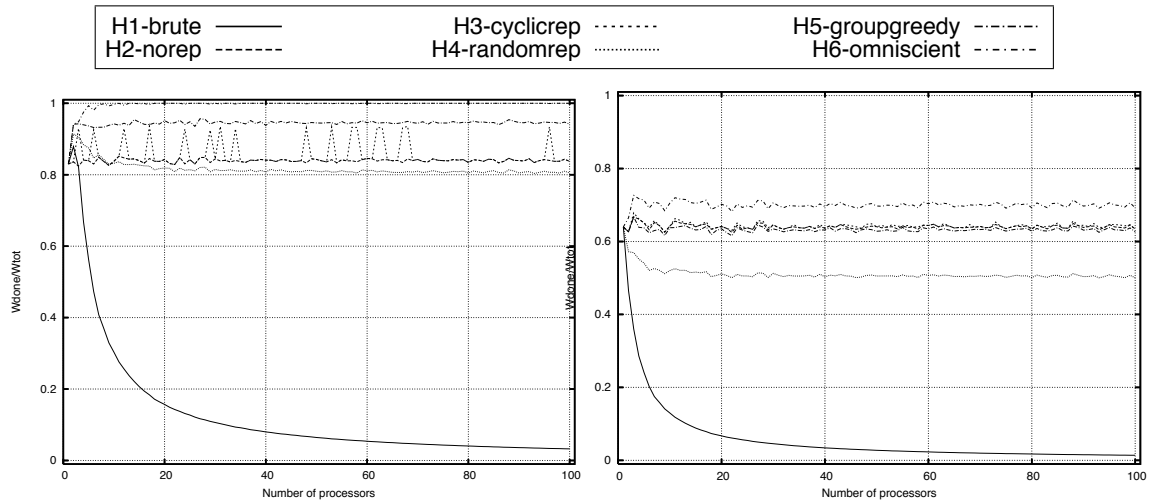
(c) 10 processors, 100 chunks.

(d) 80 processors, 100 chunks.

(e) 10 processors, 997 chunks.

(f) 80 processors, 997 chunks.

Figure 8: (E1) with $\varepsilon = 0.00001$, varying W_{tot}.



(a) $\alpha = 0.3, n = 50, \varepsilon = 0.00001$.

(b) $\alpha = 0.7, n = 50, \varepsilon = 0.00001$.

(c) $\alpha = 0.3, n = 250, \varepsilon = 0.00001$.

(d) $\alpha = 0.7, n = 250, \varepsilon = 0.00001$.

(e) $\alpha = 0.3, n = 50, \varepsilon = 0.001$.

(f) $\alpha = 0.7, n = 250, \varepsilon = 0.001$.

Figure 9: (E2): varying p.

It is less obvious in the intermediate case of $\varepsilon = 0.001$ (Figures 10(c) and 10(d)), but even in this case, the performance decreases when the number of chunks is more important. Of course, care should be taken about the exact number of chunk if using H3-cyclicrep whose performance fluctuates, as observed before.

When there are more processors in the system, the influence of the start-up cost is less important; for instance in the case of a big workload, even with $\varepsilon = 0.01$, one can use up to 1000 chunks per unit of work with no decrease in the performance (Figure 11(f)).

However, for all settings, if the chunk size becomes significantly smaller than the start-up cost, the performance of all heuristics decreases.

(E4) Fixed W_{tot}, p, and cs

In this last set of experiments, we study the impact of the start-up cost on the solution. Values of ε are taken between 0 and 1. We consider values of $n = 10, 100, 500$.

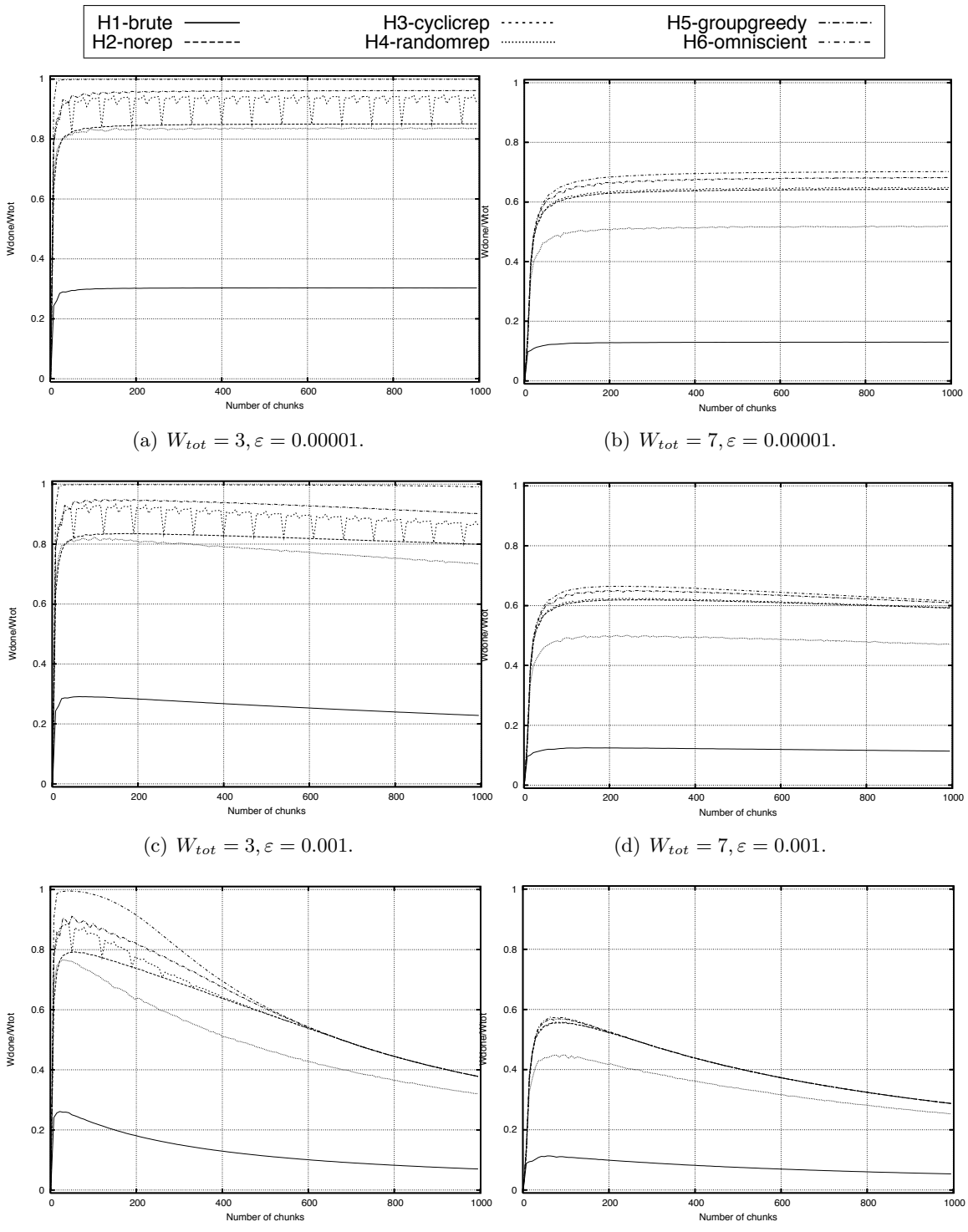
First we run simulations with $p = 10$ processors (see Figure 12). In this case, as soon as the chunks are small enough ($n \geq 100$), the start-up cost has an important effect on the result, and the performance of all heuristics decreases drastically with the increase of ε. Moreover, as soon as the start-up cost is large enough, all heuristics (except H1) find the optimal solution. There is still a difference only in the case $W_{tot} = 1$ and $n = 100$ (Figure 12(c)) and for $n = 10$ (Figures 12(a) and 12(b)). These plots confirm the ranking of the different heuristics: H4-randomrep is better than H2-norep and H3-cyclicrep for small values of ε in Figure 12(a), and the group heuristic H5-groupgreedy is most of the time better than the three other heuristics. Note that H4-randomrep has a very random behaviour which was not observed before.

Results are quite similar when tackling a platform with $p = 50$ processors (see Figure 13). Due to the small workload $W_{tot} = 1$ on Figures 13(a), 13(c) and 13(e), the whole work can be replicated onto all processors, using different permutations of chunks, and the difference between the different heuristics can be observed. With the increase of the workload ($W_{tot} = 10, 30$) the start-up cost has less impact on the performance of the heuristics.

6.3 Summarizing the Experiments

The experiments have confirmed the fact that all of our group heuristics implement an efficient way to distribute chunks. In many cases, H5-groupgreedy (Θ_{greedy}) provides the optimal solution. However, we observe that for some values of the parameters, H5-groupgreedy performs only slightly better than simpler heuristics such as H3-cyclicrep. For heavily loaded systems, a simple cyclic distribution of work can even be better than any sophisticated group heuristic, because we cannot ensure that the number of chunks will always be a multiple of g, and this cardinality may not even be fixed within a large computation: there may be groups with different values, g or $\mathsf{g} - 1$.

These experiments have also showed the impact of the start-up cost on the solution; in particular, the simulations allow us to determine how many chunks should be used in order to maximize the work done, for a given setting. If the start-up cost is negligible, small chunks can be used, but this strategy becomes disadvantageous with big start-up costs.



(a) $W_{tot} = 3, \varepsilon = 0.00001$.

(b) $W_{tot} = 7, \varepsilon = 0.00001$.

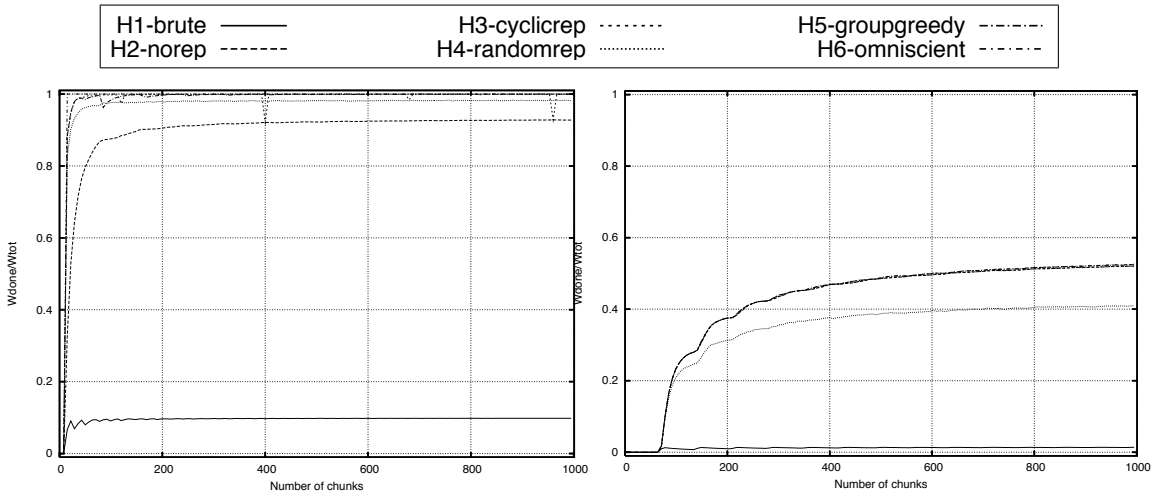
(c) $W_{tot} = 3, \varepsilon = 0.001$.

(d) $W_{tot} = 7, \varepsilon = 0.001$.

(e) $W_{tot} = 3, \varepsilon = 0.01$.

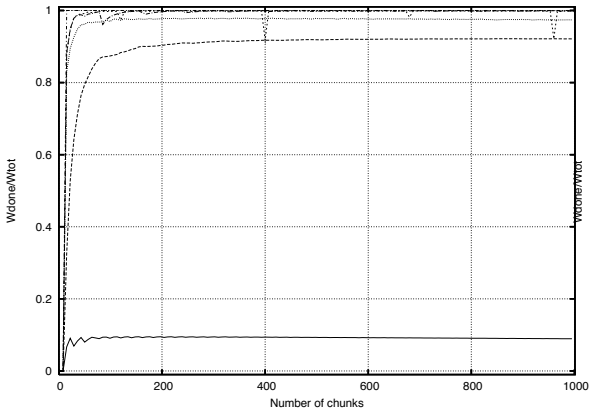
(f) $W_{tot} = 7, \varepsilon = 0.01$.

Figure 10: (E3) with 10 processors: varying cs.

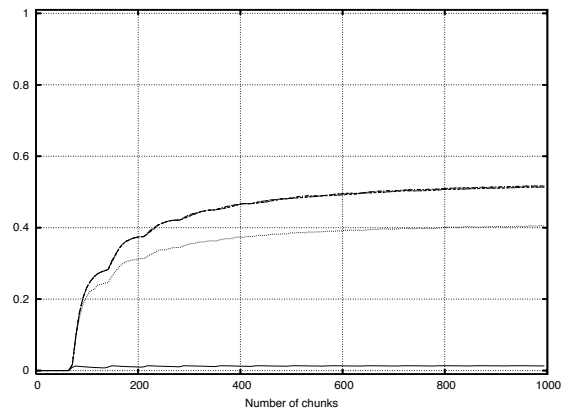


(a) $W_{tot} = 10, \varepsilon = 0.00001$.

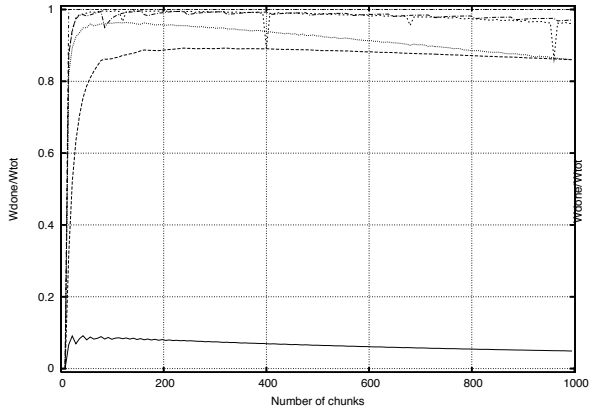
(b) $W_{tot} = 70, \varepsilon = 0.00001$.



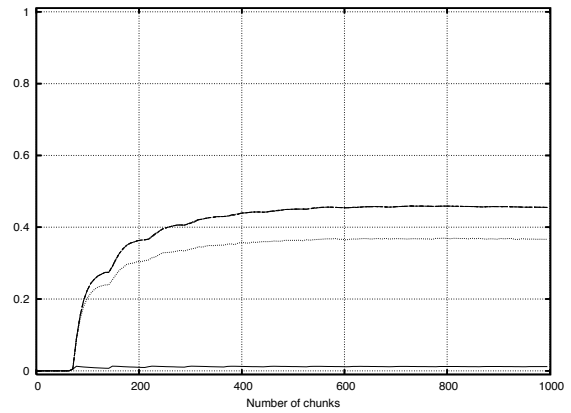
(c) $W_{tot} = 10, \varepsilon = 0.001$.



(d) $W_{tot} = 70, \varepsilon = 0.001$.

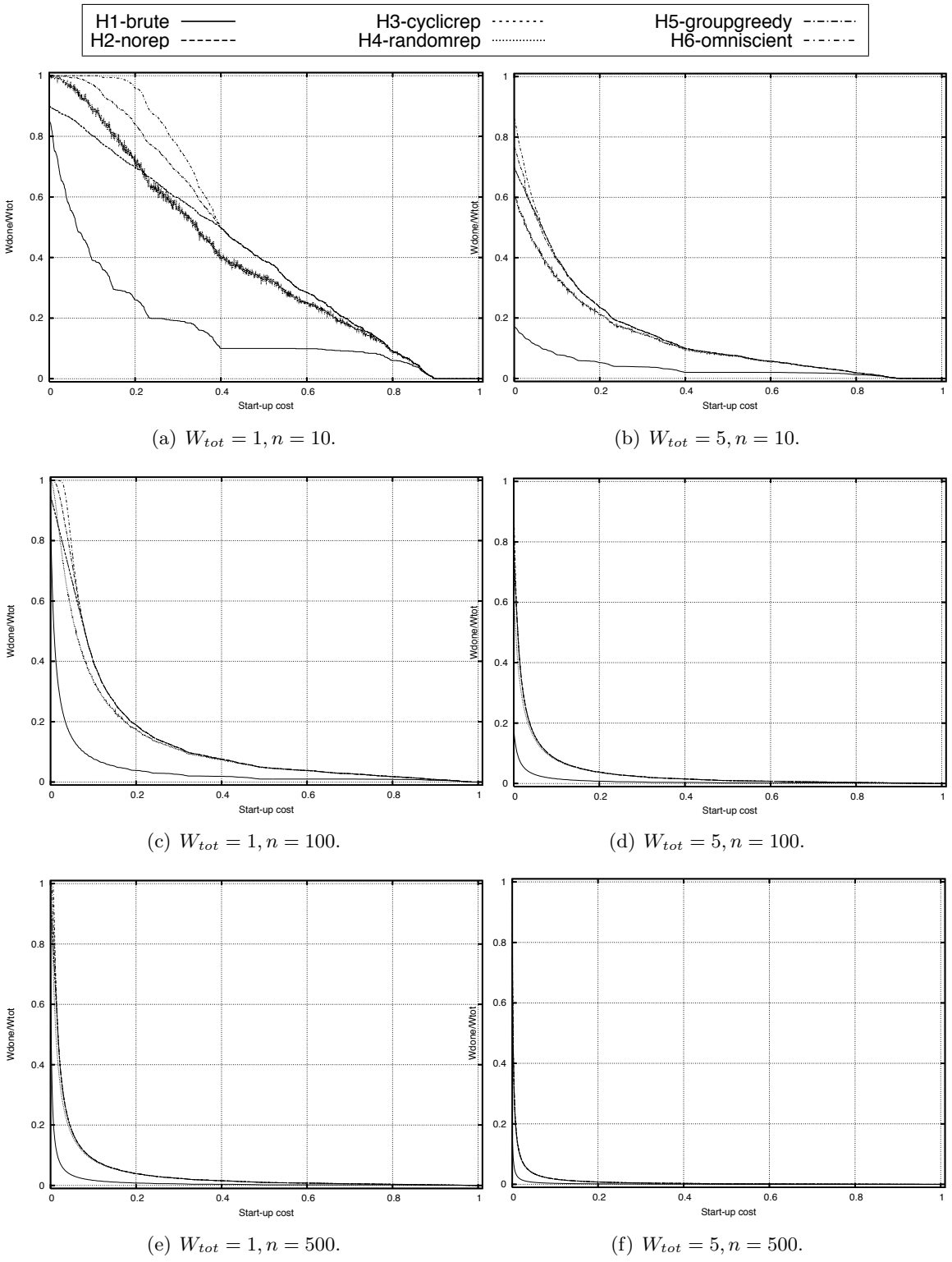


(e) $W_{tot} = 10, \varepsilon = 0.01$.



(f) $W_{tot} = 70, \varepsilon = 0.01$.

Figure 11: (E3) with 80 processors: varying cs.

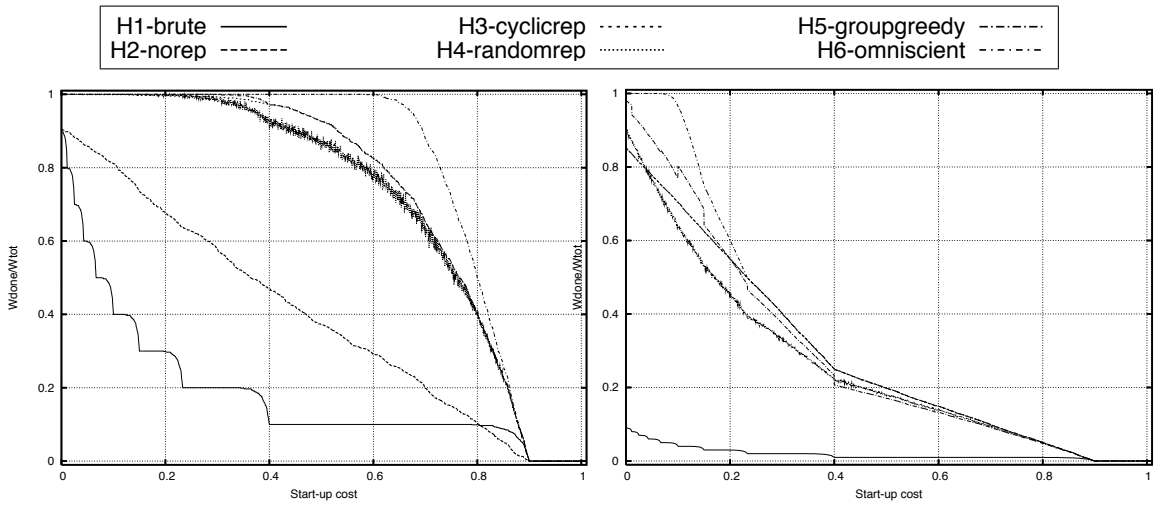


(a) $W_{tot} = 1, n = 10$. (b) $W_{tot} = 5, n = 10$.

(c) $W_{tot} = 1, n = 100$. (d) $W_{tot} = 5, n = 100$.

(e) $W_{tot} = 1, n = 500$. (f) $W_{tot} = 5, n = 500$.

Figure 12: (E4) with 10 processors: varying ε.



(a) $W_{tot} = 1, n = 10$.

(b) $W_{tot} = 10, n = 10$.

(c) $W_{tot} = 1, n = 100$.

(d) $W_{tot} = 10, n = 100$.

(e) $W_{tot} = 1, n = 500$.

(f) $W_{tot} = 30, n = 10$.

Figure 13: (E4) with 50 processors: varying ε.

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 (Section 2). Our goal has been to find 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 the results we report assume that the risk of a remote computer's being interrupted increases *linearly* with the amount of time that the computer has been available to us; a few results provide scheduling guidelines for more general risks.

We have completely solved this scheduling problem for the case of $p = 1$ remote computer (Section 3). Our solution provides exactly optimal schedules—whose expected work completion is exactly maximum—both for the free-initiation model, wherein checkpointing incurs no overhead, and the charged-initiation model, wherein checkpointing does incur an overhead. For the case of $p = 2$ remote computers, we provide schedules whose expected work completion is asymptotically optimal, as the size of the workload grows without bound; we also provide some guidelines for deriving exactly optimal schedules (Section 4). The complexity of the development in Section 4 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 (Section 5). Simulations suggest that one of our six heuristics—regrettably, the computationally most complicated one—is the clear winner in terms of performance. An extensive suite of simulation experiments suggests that all of our heuristics provide schedules with good expected work output, and that the "clear winner" in the competition of Section 5 does, indeed, dominate the others (Section 6).

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 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] M. Adler, Y. Gong, A.L. Rosenberg (2003): Asymptotically optimal worksharing in HNOWs: how long is "sufficiently long?" *36th Ann. Simulation Symp.*, 39–46.

[2] M. Adler, Y. Gong, A.L. Rosenberg (2008): On "exploiting" node-heterogeneous clusters optimally. *Theory of Computing Systs. 42*, 465–487.

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

[4] M. Banikazemi, V. Moorthy, D.K. Panda (1998): Efficient collective communication on heterogeneous networks of workstations. *Intl. Conf. on Parallel Processing*, 460–467.

[5] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert (2004): Scheduling strategies for master-slave tasking on heterogeneous remote computer grids. *IEEE Trans. Parallel and Distr. Systs. 15*, 319–330.

[6] G.D. Barlas (1998): Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary remote computer trees. *IEEE Trans. Parallel and Distr. Systs. 9*, 429–441.

[7] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert (2002): Bandwidth-centric allocation of independent tasks on heterogeneous platforms. *Intl. Parallel and Distr. Processing Symp.*

[8] O. Beaumont, A. Legrand, Y. Robert (2003): The master-slave paradigm with heterogeneous processorss. *IEEE Trans. Parallel and Distr. Systs. 14*, 897–908.

[9] O. Beaumont, L. Marchal, Y. Robert (2005): Scheduling divisible loads with return messages on heterogeneous master-worker platforms. *High-Performance Computing: The 12th Intl. Conf. Lecture Notes in Computer Science 3769*, Springer-Verlag, Berlin, 498–507.

[10] M.A. Bender and C.A. Phillips (2007): Scheduling DAGs on asynchronous processors. *19th ACM Symp. on Parallel Algorithms and Architectures*, 35–45.

[11] V. Bharadwaj, D. Ghose, V. Mani (1994): Optimal sequencing and arrangement in distributed single-level tree networks. *IEEE Trans. Parallel and Distr. Systs. 5*, 968–976.

[12] V. Bharadwaj, D. Ghose, V. Mani (1995): Multi-installment load distribution in tree networks with delays. *IEEE Trans. Aerospace and Electronic Systs. 31*, 555–567.

[13] V. Bharadwaj, D. Ghose, V. Mani, T.G. Robertazzi (1996): *Scheduling Divisible Loads in Parallel and Distributed Systems*. J. Wiley & Sons, New York.

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

[15] S.-S. Boutammine, D. Millot, C. Parrot (2006): Dynamically scheduling divisible load for grid computing. *High-Performance Computing and Communications*. In *LNCS 4208*, Springer-Verlag, Berlin, 763–772.

[16] R. Buyya, D. Abramson, J. Giddy (2001): A case for economy Grid architecture for service oriented Grid computing. *10th Heterogeneous Computing Wkshp.*

[17] Y.C. Cheng and T.G. Robertazzi (1990): Distributed computation for tree networks with communication delays. *IEEE Trans. Aerospace and Electron. Systs. 26*, 511–516.

[18] W. Cirne and K. Marzullo (1999): The Computational Co-Op: gathering clusters into a metacomputer. *13th Intl. Parallel Processing Symp.*, 160–166.

[19] P.-F. Dutot (2003): Master-slave tasking on heterogeneous processors. *17th Intl. Parallel and Distributed Processing Symp.*

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

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

[22] P. Fraigniaud, B. Mans, A.L. Rosenberg (2005): Efficient trigger-broadcasting in heterogeneous clusters. *J. Parallel and Distributed Computing 65* (2005) 628–642.

[23] L. Gao and G. Malewicz (2007): Toward maximizing the quality of results of dependent tasks computed unreliably. *Theory of Computing Systs. 41*, 731–752.

[24] M. Harchol-Balter and A. Downey (1996): Exploiting process lifetime distributions for dynamic load balancing. *SIGMETRICS'96*, 13–24.

[25] R. Kesavan, K. Bondalapati, D.K. Panda (1996): Multicast on irregular switch-based networks with wormhole routing. *3rd Intl. Symp. on High-Performance Computer Architecture.*

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

[27] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky (2000): SETI@home: massively distributed computing for SETI. In *Computing in Science and Engineering* (P.F. Dubois, Ed.) IEEE Computer Soc. Press, Los Alamitos, CA.

[28] M. Litzkow, M. Livny, M.W. Mutka (1988): Condor – A hunter of idle workstations. *8th Intl. Conf. on Distr. Computing Systs.*, 104–111.

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

[30] G.F. Pfister (1995): *In Search of Clusters.* Prentice-Hall.

[31] A.L. Rosenberg (1999): Guidelines for data-parallel cycle-stealing in networks of workstations, I: on maximizing expected output. *J. Parallel and Distr. Comput. 59*, 31–53.

[32] A.L. Rosenberg (2000): Guidelines for data-parallel cycle-stealing in networks of workstations, II: on maximizing guaranteed output. *Intl. J. Foundations of Computer Science 11*, 183–204.

[33] A.L. Rosenberg (2001): On sharing bag-of-tasks workloads in heterogeneous networks of workstations: greedier is not better. *3rd IEEE Intl. Conf. on Cluster Computing*, 124–131.

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

[35] A.L. Rosenberg (2006): Changing challenges for collaborative algorithmics. In *Handbook of Nature-Inspired and Innovative Computing: Integrating Classical Models with Emerging Technologies* (A. Zomaya, ed.) Springer, New York, pp. 1–44.

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

[37] J. Wingstrom and H. Casanove (2008): Probabilistic allocation of tasks on desktop grids. *Wkshp. on Large-Scale, Volatile Desktop Grids (PCGrid'08)*.

[38] Y. Yang and H. Casanova (2003): UMR: A multi-round algorithm for scheduling divisible workloads. *17th Intl. Parallel and Distributed Processing Symp.*