Internet Computing of Tasks with Dependencies Using Unreliable Workers*

(Extended Abstract)

Li Gao and Grzegorz Malewicz

University of Alabama, Tuscaloosa, AL 35487, USA
{lgao, greg}@cs.ua.edu

Abstract. This paper studies the problem of improving the effectiveness of computing dependent tasks over the Internet. The distributed system is composed of a reliable server that coordinates the computation of a massive number of unreliable workers. It is known that the server cannot always ensure that the result of a task is correct without computing the task itself. This fact has significant impact on computing interdependent tasks. Since the computational capacity of the server may be restricted and so may be the time to complete the computation, the server may be able to compute only selected tasks, without knowing whether the remaining tasks were computed by workers correctly. But an incorrectly computed task may render the results of all dependent tasks incorrect. Thus it may become important for the server to compute judiciously selected tasks, so as to maximize the number of correct results.
In this work we assume that any worker computes correctly with probability $p < 1$. Any incorrectly computed task corrupts all dependent tasks. The goal is to determine which tasks should be computed by the (reliable) server and which by the (unreliable) workers, and when, so as to maximize the expected number of correct results, under a constraint d on the computation time. We show that this optimization problem is NP-hard. Then we study optimal scheduling algorithms for the mesh with the tightest deadline. We present combinatorial arguments that completely describe optimal solutions for two ranges of values of worker reliability p, when p is close to zero and when p is close to one.

1 Introduction

This paper begins developing a scheduling theory for improving the quality of results of tasks executed unreliably over the Internet. We introduce a combinatorial optimization problem, show that the problem is NP-hard, and then study the problem restricted to the mesh where we give optimal polynomial time algorithms.

* Contact author: Grzegorz Malewicz, Department of Computer Science, University of Alabama, 116 Houser Hall, Tuscaloosa, AL 35487-0290, USA, Phone (205) 348-4038, Fax (205) 348-0219.

T. Higashino (Ed.): OPODIS 2004, LNCS 3544, pp. 443–458, 2005.

There is a large number of underutilized computers connected to the Internet. Harnessing their power can enable the creation of a distributed supercomputer that can accomplish sheer volumes of work at a tiny fraction of the cost of a more traditional, more centralized, supercomputer. Several successful implementations of Internet Supercomputers exist today [8, 11, 16, 21]. These enable solving problems composed of a large number of *tasks*. In these implementations, a computer, called a *server*, allows any other computer, called a *worker*, to register and download a piece of software. Then the software requests a task from the server, downloads appropriate data that describe the task, executes the task, and returns the result to the server. This process repeats. When the number of workers is large, the Internet Supercomputer achieves high computing speed. For example, the SETI@home project reported its speed to be 57.29 Teraflops [23]. These Internet Supercomputers are also called Internet or Web Computing platforms, or High-Throughput Computing Grids [4].

SETI@home stated [9] that half of the resources of the project were spent on dealing with security problems. One of them is to ensure the quality of results returned by workers. Some computers, such as the server and perhaps certain workers, are reliable; they will correctly execute the tasks assigned by the server. However, workers are commonly unreliable. That is they may return to the server incorrect results due to unintended failures caused for example by overclocked processors, or they may deceivingly claim to have performed assigned work so as to obtain incentives such as getting higher rank on the SETI@home list of contributed units of work. Several schemes were proposed to improve the quality of results of tasks. The schemes encompass modeling reliability of workers based on the history of interaction with workers [10, 24], keeping track of which task was assigned to which worker [20], sending the same task to multiple workers [11], and verifying results returned by workers [6, 7, 25, 3].

It appears that in general it is fundamentally difficult to develop a method that ascertains that a task was executed correctly, without the task being executed on a reliable computer (cf. [1]). This difficulty has significant consequences on performing a computation described by a directed acyclic graph. Individual tasks of a computation may be quite computationally intensive, and so it may be unrealistic to execute too many of them on reliable computers that may be scarce. Consequently, the server may not always know if a given task was executed correctly or not. When tasks have dependencies, and the result of a task is incorrect, then all descendant tasks, even if executed correctly, may produce incorrect results, simply because their input depends, directly or indirectly, on the result of the one task that was executed incorrectly. It seems plausible that in such setting some tasks may have high impact on the total number of correct results, while other tasks may have low impact. The possible asymmetry means that it may be important to judiciously select which tasks should be executed reliably, and which other tasks can be left to unreliable workers, so as to maximize the total number of correct results.

This paper begins developing a scheduling theory for increasing the number of correct results of tasks executed on unreliable workers, when tasks have de-

pendencies. Let us discuss some tradeoffs. A naive scheduling approach would be to execute all tasks on reliable computers only. Of course, then there is no need for judicious selection at all. However, the number of reliable computers may be quite small compared to the number of unreliable computers. Therefore, when work is assigned to reliable computers only, relatively more time would be needed to complete the entire computation. We could reduce the computation time by including unreliable computers in the computation, at the cost of reducing the number of correct results. Thus we expect that the number of correct results, for a given directed acyclic graph that describes dependencies between tasks, is related to two parameters: the reliability of computers and the deadline to complete the computation. Our ultimate goal is to fully understand this relationship.

One natural way to model unreliability of computers is to assume a probabilistic setting. Each computer will execute correctly with a certain probability. This assumption could be justified, for example, by the fact that one source of computation errors in Internet Supercomputers is overclocked processors [9].

Towards this end we formulate a model of an Internet Supercomputer. Our model extends the Internet pebble game introduced recently by Rosenberg [19]. The computation is modeled by a finite directed acyclic graph. Each node in the dag represents a task. There is an unbounded number of computers in the system. Computer i executes a task *correctly* with probability p_i and *incorrectly* with probability $1-p_i$. This probability is called *reliability* of the computer. There are three types of *pebbles* used to play the game. Initially all sources of the dag are pebbled with an *eligible* pebble. At any discrete time t we select a computer, say i, and a task that has an eligible pebble. Then we replace the eligible pebble with a pebble *executed correctly* with probability p_i, and with a pebble *executed incorrectly* with probability $1 - p_i$. Any task that does not have any pebble but all its parents have executed pebbles is pebbled with an eligible pebble. Any task that is executed incorrectly *corrupts* the results of all descendant tasks; so their results will be incorrect even if executed correctly. There is a *deadline* d by which all tasks of the dag must be executed. The goal is to determine which computer should execute which task and when, so as to maximize the expected number of correct results. Solving this optimization problem is important. One would like to establish theoretical guidelines for how to effectively and quickly execute a computation composed of dependent tasks, using unreliable computers.

The focus of this paper is to study a specific version of the scheduling problem. We consider a dag called a (two dimensional) mesh that is composed of k^2 nodes arranged into k rows and k columns. Each node has an arc to the node in the next column (if it exists) of the same row, and an arc to the node in the next row (if it exists) of the same column. Our choice of a mesh is motivated by the fact that meshes are a convenient way to structure computation and they arise in practice (cf. [19]). We investigate how to compute the mesh as quickly as possible i.e., we fix deadline d to $2k-1$. We assume that there is a single computer whose reliability is 1; this computer is called the server. Any other computer has reliability $0 < p < 1$; this computer is called a worker. Our assumption that

there is a single reliable server and each worker has the same reliability p seems to be a natural "first approximation" of an Internet Supercomputer composed of unreliable workers.

We note that even if results of tasks cannot be ascertained to be correct in general without computing the tasks on trusted computers, one could still compute tasks redundantly on unreliable computers and use majority voting hoping to improve the quality of results. Such an approach is orthogonal to the aim of this paper where we want to use as little resources as possible (which is demonstrated by the fact that each task is computed by a single computer), while still obtaining as much quality as possible by judiciously assigning computers to tasks.

Contributions. This paper begins developing a scheduling theory for maximizing the number of correct results of tasks with dependencies executed unreliably over the Internet. Our specific contributions are as follows:

(a) We introduce a probabilistic pebble game that models internet computing with unreliable workers, and a new combinatorial optimization problem.
(b) We show that the optimization problem is NP-hard by a chain of reductions from the Balanced Complete Bipartite Subgraph Problem. In fact the problem is NP-hard even when restricted to bipartite dags computed by a single (reliable) server and (unreliable) workers.
(c) We give polynomial time optimal scheduling algorithms for the mesh under the tightest deadline $d = 2k-1$, that use a server and workers, where worker reliability p falls into two ranges of values. We show that expectation is maximized when tasks are executed roughly in breadth-first search order, and the server executes exactly one task per "level" of the mesh. We demonstrate that there are two scheduling regimes. These regimes depend on the value of reliability p. The first regime is when reliability is close to 1. We completely characterize maximal schedules in this regime. The server should execute a "central" task at any time. Specifically, at any time, there is some number of "eligible" tasks that can be executed given task precedence constraints and tasks executed so far. These tasks form a diagonal "level" of the mesh. At this time, the server should execute a task that has the most descendants from among these eligible tasks, which turns out to be a central task on the diagonal level (there may be two such tasks, in which case the choice does not matter, as we show), and workers should execute all other eligible tasks. Intuitively, is appears that when p is close to 1, then optimal schedules are "descendant driven". The second regime is when reliability is close to 0. We also completely characterize maximal schedules in this regime. The server should execute an "edge" task at any time. Specifically, the server should either execute tasks in the top row and the rightmost column, or it should execute tasks in the leftmost column and the bottom row. Intuitively, is appears that when p is close to 0, then optimal schedules are "ancestor driven". The demonstration that there are two distinct regimes is, we believe, an important contribution of this paper that indicates that the problem has a non-trivial and interesting structure of optimal solution (that we begin to explore).

Paper organization. The rest of the paper is structured as follows. In Section 2, we present a model of Internet Supercomputing with unreliable workers, and formulate an optimization problem of maximizing the expected number of correct results of tasks. In Section 3, we show that the optimization problem is NP-hard. Then, in Section 4, we give polynomial time optimal scheduling algorithms for a mesh. Next, in Section 5, we discuss related work. Finally, in Section 6, we conclude and discuss future work. Due to space limitations most proofs are omitted from this extended abstract.

2 Definitions and Preliminaries

A directed acyclic graph $G = (V, E)$, or dag for short, on n nodes abstracts *computation* composed of tasks and information flow between tasks (all dags are finite in this paper). We often refer to the nodes as tasks. A *path* is a sequence $u_1, u_2, \ldots, u_k$ of two or more nodes such that there is an arc from u_i to u_{i+1}, for $1 \leq i \leq k-1$. In a dag no such path can have $u_1 = u_k$. For a given node u, $P(u)$ is the set of *parents* of u i.e., of all nodes v, such that there is an arc from v to u; $C(u)$ is the set of *children* of u i.e., of all nodes v, such that there is an arc from u to v; $A(u)$ is the set of *ancestors* of u i.e., of all nodes v, such that there is a path from v to u; and $D(u)$ is the set of *descendants* of u i.e., of all nodes v such that there is a path from u to v. Note that $u \notin P(u)$, $u \notin C(u)$, $u \notin D(u)$, and $u \notin A(u)$. A task u such that $P(u) = \emptyset$ is called a *source*, and when $C(u) = \emptyset$ then u is called a *sink*.

A *schedule* describes when tasks are executed and by whom. A schedule has two components. The first component is a function x that takes a natural number $t \geq 1$ and returns the subset of tasks $x(t)$ that are executed at time t. There is a $\mu \geq 1$ such that each set $x(1), \ldots, x(\mu)$ is not empty, and the sets partition the set of all tasks. The number μ is called *makespan* of the schedule. Execution of any task takes one unit of time. A task can only be executed when all its ancestors already have, so for any $1 \leq t \leq \mu$, $x(t)$ must be a subset of tasks whose ancestors are in $x(1) \cup \ldots \cup x(t-1)$. The second component is a function c that takes a natural number v and returns a number $c(v)$ denoting the *computer* that executes task v. We assume that there are m computers in the system. It must be the case that any computer executes at most one task per unit of time, so for any $1 \leq t \leq \mu$, and any i, the number $\left|c^{-1}(\{i\}) \cap x(t)\right|$ of tasks executed by computer i at time t is at most one. For any dag there is at least one schedule (x, c).

Computers are *unreliable*. When a computer i executes a task, then with probability p_i the computer executes u *correctly*, independently from the execution of other tasks. However, with probability $1 - p_i$ the computer executes the task *incorrectly*. We call p_i the *reliability* of the computer. Such incorrect execution affects the results of every task in $D(u)$. Intuitively, an incorrectly executed task u corrupts the results of any descendant task v, because the result of u is used, directly or indirectly, when the task v is executed. We say that the *result of a task is correct*, if the task and all its ancestors are executed correctly. In

contrast, the result of a task is incorrect, if either the task or one of its ancestors is executed incorrectly.

We can compute the expected number of correct results for a given schedule (x, c). In order for a task u to be computed correctly, every task in $A(u) \cup \{u\}$ must be computed correctly. The function c defines which computer executes each of these tasks. So by independence, the probability that the result of u is correct is the product $\prod_{v \in A(u) \cup \{u\}} p_{c(v)}$. Let E_u be the indicator random variable equal to 1 if the result of task u is correct, and 0 otherwise. Then the total number of correct results is equal to $E = \sum_{u \in V} E_u$. By linearity of expectation

$$\mathsf{Exp}\,[E] = \sum_{u \in V} \mathsf{Exp}\,[E_u] = \sum_{u \in V} \prod_{v \in A(u) \cup \{u\}} p_{c(v)} \ .$$

Our goal is to find a schedule (x, c) that maximizes this expectation.

Constrained Computing with Unreliable Workers
Instance: A dag G that represents tasks and information flow between them, a deadline d, and m computers with reliabilities $p_1, \ldots, p_m$.
Objective: Find a schedule (x, c) with makespan at most d that maximizes the expected number of correct results.

This paper focuses on the case where there is a single computer, called the server, with reliability 1, and any other computer, called worker, has reliability $0 < p < 1$. In this case our optimization problem has a simpler formulation. Suppose that R is the subset of tasks that the server executes. We call this subset a *server subset*. Let $E(R)$ be the random variable equal to the number of correct results for a schedule with the set R of tasks executed by the server. Then the expected number of correct results is equal to

$$\mathsf{Exp}\,[E(R)] = \sum_{u \in V} p^{|(A(u) \cup \{u\}) \setminus R|} \ .$$

Note that this expectation depends on the graph G and the set R, but does not depend on the sequence x in which tasks have been executed, nor if the deadline constraint has been violated or if the server executed more than one task at a time. Trivially, the expectation is maximized when all tasks are executed by the server, $R = V$. However, then it may happen that either the makespan of the schedule is large, or there is a time when the server is supposed to execute many tasks. We are looking for a server subset R that maximizes the expectation, and a function x, such that at most one task is executed by the server at any point of time in x and makespan of x is at most d. We refer to this restricted version of the problem as Internet Supercomputing with Unreliable Workers (ISUW).

3 Complexity of the Problem

We demonstrate that it is NP-hard to solve the problem of Internet Supercomputing with Unreliable Workers. The proof is composed of two steps. We first reduce a known NP-complete problem called Balanced Complete Bipartite Subgraph Problem (see [5] problem GT24) to an "intermediate" problem of selecting subsets whose union is small. Then we show how to give answer to any instance of the intermediate problem using an algorithm that finds a solution to the problem of Internet Supercomputing with Unreliable Workers. This will immediately imply that Constrained Computing with Unreliable Workers is also NP-hard.

Many Subsets with Small Union (MSSU)
Instance: Nonempty subsets $S_1, \ldots, S_n$ of $[n]$, such that their union is $[n]$, and numbers $a \leq n$ and $b \leq n$.
Question: Can a of these subsets be selected whose union has cardinality at most b?

We give a reduction is from the Balanced Complete Bipartite Subgraph Problem (BCBS) (see [5] problem GT24, and [17] for recent results and references) to MSSU.

Lemma 1. *The Many Subsets with Small Union Problem is NP-complete.*

Proof. The reduction is from the Balanced Complete Bipartite Subgraph Problem (BCBS) (see [5] problem GT24, and [17] for recent results and references). Recall that in the problem we are given a bipartite graph and a number k and we want to know if the graph contains an induced complete bipartite subgraph with k nodes on the left and k on the right.

Let us take any bipartite graph G on $n-1$ nodes and a k. Consider an expanded graph G' with one extra node n that is isolated. Naturally, G' is also a bipartite graph. Observe that there is a balanced complete bipartite subgraph with k nodes on the left and k on the right in G, if and only if there is such a subgraph in G' (the isolated node in G' cannot belong to the subgraph).

We now define an instance of the Many Subsets with Small Union Problem. Let M be the complement of the adjacency matrix of the graph G'. Note that the bottom row n and the right-most column n are filled with ones, because of the isolated node. We define the set S_i, $1 \leq i \leq n$, so that the characteristic vector of the set is equal to the column i of M. So each S_i is nonempty (because of the bottom row) and their union $S_1 \cup \ldots \cup S_n$ is exactly $[n]$ (because of the right-most column). Let $b = n - k$ and $a = k$.

The graph G' has a balanced complete bipartite subgraph on $2k$ nodes, if and only if we can rearrange rows and columns of M so that the top left k by k square of the rearranged M has zeros only. But this can be done if and only if we can select $a = k$ of the subsets, so that the union of the selected subsets has cardinality at most $b = n - k$. This completes the proof.

We then give a polynomial time Turing transformation from MSSU to ISUW. In our transformation we construct a bipartite dag with sets associated with sinks and elements associated with sources.

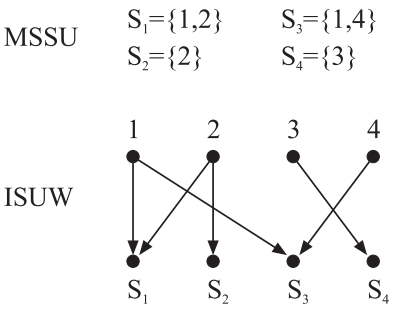

Fig. 1. Reduction

Theorem 1. *The Internet Supercomputing with Unreliable Workers Problem is NP-hard.*

Proof. We take any instance of the MSSU Problem and show how to answer the question posed in the problem, using an algorithm that maximizes expectation for instances of the ISUW Problem.

Let $S_1, \ldots, S_n$ be any nonempty subsets such that $S_1 \cup \ldots \cup S_n = [n]$ and a and b be numbers at most n. We construct an instance of the ISUW Problem. The dag describes subset membership. It has two "levels": the "bottom" tasks correspond to sets, while the "top" tasks correspond to elements of $[n]$. So each level has n tasks, and the total number of tasks in the dag is $2n$. We place an arc from a top task i to a bottom task j, if element i is in subset S_j. See Figure 1 for an example of the dag constructed for given subsets. We notice that each top task is linked to at least one bottom task, because the union of the subsets is $[n]$. Moreover, each bottom task is linked to a top task, because each subset is nonempty. We define the deadline to be $d = b + 1 \leq 2n$, and reliability of a worker to be $p = 1/n^2$. Let R be a server subset that maximizes the expected number of correct results under the constraints.

We argue that any maximum solution, including R, must have a special structure. First, $|R| = d$. Indeed, the cardinality of R cannot be larger, because then the deadline constraint would be violated, and it cannot be smaller, because then expectation could be strictly increased by executing one more task on the server without violating constraints. Second, at least one task of R belongs to the bottom level. Indeed, if R had d tasks on the top level, then one of them would be executed at time d or later. But this task has a child, and so this child could only be executed at time $d + 1$ or later, thus violating the deadline constraint. Third, for similar reasons, at least one task from R must be on the top level. These three observations imply that any server subset that maximizes expectation has cardinality d and has at least one task on the top and at least one on the bottom level. Note that for any server subset with these properties, there is a trivial way to execute the $2n$ tasks under the constraints. The server subsets constructed in the remainder of the proof will have these properties, and so we do not explicitly construct functions x in the remainder of the proof.

There must be a server subset R' with the same expectation as R, such that exactly $d-1$ of the tasks from R' belong to the top level. Indeed, we demonstrate that as long as there are two tasks in R that belong to the bottom level, we can remove a bottom task from R and add a new top task to R without decreasing expectation. Thus we can keep on removing and adding tasks until exactly one task from R belongs to the bottom level, never reducing expectation. Suppose that there are two distinct tasks v and w from R that belong to the bottom level. Since $d \leq n+1$ and two tasks from R are at the bottom level, then there is a task u at the top level that is not in R. Let R' be a server subset equal to R except that v is excluded but u is included instead. Clearly, R' has cardinality d and has at least one task at the bottom and at least one task at the top level. It remains to be seen that R' has no smaller expectation. In R, u contributed p to the expectation and v contributed p^h, for some $h \geq 0$ (when v has parents only among R then $h = 0$). In R', however, u contributes 1 and v contributes at least $p^h \cdot p$. In addition, the contribution of any task other than v that has u as a parent will increase as well. No other task will change its contribution. So the difference in expectation is at least $\left(1 + p^h \cdot p\right) - \left(p + p^h\right) = (1-p) - p^h\,(1-p) \geq 0$. In fact this difference must be exactly 0, because R is maximal. Thus expectation for R' is the same as expectation for R.

Suppose that maximum expectation is z. We shall see that by inspecting z, we can answer whether there are a subsets in the instance of the MSSU Problem, such that the union of these subsets has cardinality at most b.

We have seen that there is a server subset R', such that the expectation for R' is the same as for R, and that $b = d - 1$ of the tasks from R' are on the top level and one is on the bottom level. Let us find out how much each task contributes to z. The remaining $n - b$ top tasks are not executed by the server. Thus the contribution of the top tasks to the expectation is $b+(n-b)/n^2$. We now study the contribution of the bottom tasks. Let us assume for a moment that no bottom task is executed by the server. If all parents of a bottom task are among the b top tasks executed by the server, then the bottom task will contribute exactly $1/n^2$; let k be the number of bottom tasks u such that the parents of u are among the b tasks, $0 \leq k \leq n$. Recall that each bottom task has a parent. So each of the remaining $n - k$ bottom tasks has a parent that is *not* among the top b tasks executed by the server. Hence such bottom task will contribute at most $1/n^4$ to the expectation. Let us now account for this one bottom task executed by the server. If $k \geq 1$, then the bottom task must be among the k tasks, because otherwise expectation could be increased by executing at the server any of k tasks instead. Thus, when $k \geq 1$, the expectation z is in the interval $[y, y + 1/n^3]$, where $y = b + (n-b)/n^2 + 1 + (k-1)/n^2$. If $k = 0$, then each bottom task has at least one parent that is not executed by the server, and so expectation z is in the interval $[y', y' + 1/n^2 + 1/n^3]$, where $y' = b + (n-b)/n^2$. Consequently, these $n + 1$ intervals, for $k = 0, 1, 2, \ldots, n$, do not overlap. Thus there exist k bottom tasks whose parents form a set of at most b tasks. Since b is known, the value of k can be determined by inspecting z. Observe also that it is not possible that there are strictly more than k bottom tasks whose parents

comprise a set of at most b top tasks, because then maximum expectation would be strictly larger than z.

Corollary 1. *The Constrained Computing with Unreliable Workers Problem is NP-hard.*

4 Optimal Algorithms for the Mesh

In this section we present optimal solutions to the scheduling problem of Internet Supercomputing with Unreliable Workers on a mesh. We fix deadline to the tightest one possible on the given mesh. Under this constraint, we completely describe the optimal solutions for two ranges of values of reliability p of workers. When the reliability is close to zero, then a server subset maximizes expectation if and only if it contains only a continuous sequence of "edge" tasks. There are two such subsets in a mesh. When the reliability is close to one, then a server subset maximizes expectation if and only if it contains only "central" tasks. There are exponentially many such subsets. The remainder of the section defines the mesh and demonstrates a basic structure of any optimal server subset. Then optimal scheduling algorithms are given for the two ranges of worker reliability. In particular, edge and central server subsets are defined, and combinatorial arguments that ascertain optimality of the subsets are presented.

4.1 Preliminaries

A *mesh* M_k, for any given $k \geq 1$, is a dag with nodes $V = \{(i,j) \mid 1 \leq i,j \leq k\}$. There is an arc from any node (i,j) to node $(i+1,j)$, as long as both nodes belong to the mesh. Similarly, there is an arc from (i,j) to $(i,j+1)$. We introduce orientation of the mesh. Specifically, the node $(1,1)$ is the North-West node, and the node (k,k) is the South-East node. See Figure 2 for an example of a mesh and its orientation. A formal definition of orientation should be clear to the reader. We use Figure 2 to refer to "left", "right" etc. A *level* ℓ is the set of nodes (i,j) of mesh M_k such that $i+j = \ell+1$. There are exactly $2k-1$ non-empty levels of M_k. The levels partition the nodes of the mesh. For any node on level ℓ, if the node has a parent, then the parent is on level $\ell-1$, if the node has a child, then the child is on level $\ell+1$. Column j is the set of nodes that have the second coordinate equal to j. Row i is the set of nodes that have the first coordinate equal to i.

We begin with a lemma that exposes a structure of an optimal solution to our restricted problem. The subsequent lemma states that any server subset that maximizes expectation must have exactly one task on each level, no more and no fewer.

Lemma 2. *For any $k \geq 1$, and the mesh M_k, let R be a server subset that maximizes the expected number of correct results and x the corresponding function such that x has makespan at most $2k-1$. Then any level ℓ, $1 \leq \ell \leq 2k-1$, shares exactly one task with R.*

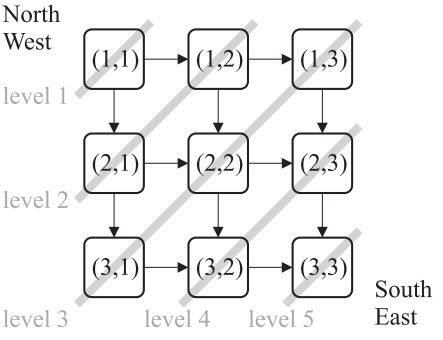

Fig. 2. Mesh M_3

This lemma considerably simplifies scheduling. Since any server subset R that maximizes expectation has exactly one task per level, we can trivially generate the function x that dictates when tasks are executed such that the deadline constraint is met. We simply schedule task execution level by level (breadth-first search order). Thus we do not explicitly construct any function x for any such server subset, keeping in mind that an appropriate x can be trivially generated.

The next question that we need to answer is: Which task should be selected on each level? We note that for any reliability p of worker, $0 < p < 1$, it is always better to execute on the server a parent of a task, instead of the task because then expectation will be strictly increased. Unfortunately, tasks on any given level are not comparable (no task is a parent of any other on the same level). Therefore, this simple observation does not help us decide which task of a given level should be executed by the server. We need a different decision algorithm instead. In the remainder of the section we present two optimal algorithms, one when p is close to 1 and the other when p is close to 0.

4.2 Optimal Algorithm for Workers with High Reliability

This section completely characterizes optimal server subsets when the reliability of worker p is close to one. We observe that then it is better when the server executes a task with more descendants. This determines which task of any odd level the server should execute. For even levels, selection is ambiguous, but we show that it does not matter, as expectation will be the same no matter how we choose.

The next lemma explains that as long as p is close enough to 1, it is better that the server executes a task with strictly more descendants. The proof observes that a task with more descendant adds strictly more to the expectation than any task with fewer descendants no matter which the other tasks are in the server subset.

Lemma 3. *Let G be a dag on n nodes, reliability p of worker be $(1 - 1/n)^{1/n} < p < 1$, u and w be two nodes such the set $D(w)$ of descendents of w has at least*

one more node than the set $D(u)$ of descendents of u, $|D(w)| \geq 1 + |D(u)|$, and R be a server subset that contains neither u nor w. Then the expected number of correct results for the server subset $R \cup \{w\}$ is strictly larger than for the server subset $R \cup \{u\}$.

Note that $(1 - 1/n)^{1/n}$ is asymptotically close to $1 - 1/n^2$.

The lemma almost settles the question for p close to 1. One can see that for any level of mesh M_k, tasks that occupy a "central" location of the level have most descendants across tasks on the level. Thus each level will have a single "central" task in an optimal R. The main issue, however, is that any even level has two "central" tasks that have the same number of descendants. The existence of these tasks makes room for ambiguity. Our next goal is to demonstrate that this ambiguity has no effect on the expected number of correct results.

For a given mesh M_k, we call a server subset R a *central server subset* if it is composed of specific tasks. It contains tasks (i, i), for any $1 \leq i \leq k$, and, in addition, for any $1 \leq i < k$, either task $(i, i+1)$ or $(i+1, i)$, but not both. Note that for any central server subset, each level of M_k contains exactly one task from the subset. We prove that expectations for central server subsets are the same by noticing that if task $(i, i+1)$ belongs to R, then we can replace the task with task $(i+1, i)$ without changing expectation.

Lemma 4. *Let R and R' be any central server subsets. Then the expected number of correct results for R is the same as it is for R'.*

We gather the observations developed so far to prove a theorem on the structure of optimal solution when p is close to 1.

Theorem 2. *Let $k \geq 1$, worker reliability $\left(1 - 1/k^2\right)^{1/k^2} < p < 1$, and deadline $d = 2k - 1$. Then a server subset S for mesh M_k maximizes the expected number of correct results if and only if S is a central server subset.*

4.3 Optimal Algorithm for Workers with Low Reliability

This section completely characterizes optimal server subsets when the reliability of worker p is close to zero. The argument has two parts. We begin by showing that any server subset that maximizes expectation must contain either all tasks of the top row or all tasks of the leftmost column. This is shown by observing that there is a tradeoff: it the first row contributes much to the expectation, then the rest of the mesh contributes little, and vice versa. A symmetric argument is applied to the leftmost column.

Lemma 5. *Let $k \geq 3$ and $0 < p \leq 1/6$. Let S be any server subset of M_k that has exactly one task per level. If S does not contain all tasks from the top row nor does it contain all tasks from the leftmost column, then S does not maximize the expected number of correct results.*

The lemma immediately implies that any optimal server subset must contain either all tasks from the top row or all tasks from the leftmost column, whenever

$0 < p \leq 1/6$ and $k \geq 3$. This settles the question which tasks from levels 1 to k must belong to an optimal server subset. What about tasks from level $k+1$ and higher? The subsequent lemma provides an inductive argument that settles this question as long as p is small. The key observation that gives rise to the proof is that the tasks on level $b+k-1$ and higher contribute little compared to the contribution that task (b,k) would make when included in a server subset.

Lemma 6. *Let $k \geq 3$, $0 < p \leq 1/(2k)$ and $2 \leq b \leq k-1$. Let S be any server subset of M_k that has exactly one task per level. If*

(i) S contains all tasks of the top row and the $b-1$ top tasks of the rightmost column, but not task (b,k), or
(ii) S contains all tasks of the leftmost column and the $b-1$ leftmost tasks of the bottom row, but not task (k,b)

then S does not maximize the expected number of correct results.

Given a mesh M_k, we call a server subset R an *edge server subset* if it is composed of specific tasks. Such subset must either contain only tasks of the top row and the rightmost column, or only tasks of the leftmost column and the bottom row. The following theorem completely characterizes the structure of optimal server subsets S for small enough worker reliability. We can prove the theorem using observations developed so far.

Theorem 3. *Let $k \geq 3$, worker reliability $0 < p \leq 1/(2k)$, and deadline $d = 2k-1$. Then a server subset S for mesh M_k maximizes the expected number of correct results if and only if S is an edge server subset.*

5 Related Work

The combinatorial optimization problem introduced in this paper is related to two known hard problems. In the Network Reliability Problem (see [5] problem ND20) we are given a graph where each edge e has a failure probability $p(e)$, a subset V' of vertices, and a number q. The goal is to decide if the probability is at least q, that for each pair of vertices in V' there is a path connecting the vertices, such that the path has no failed edges. Our problem is similar, because we check if all ancestors have been executed correctly (which resembles checking for the existence of paths with no failed edges from the node to all ancestors). However, our optimization goal is different, as we count the number of nodes for which all ancestors have not failed. In a different problem, called the Network Survivability Problem (see [5] problem ND21), we are given a graph where each edge and each node has a failure probability, and a number q. The goal is to find out if the probability is at least q that for all edges $\{u,v\}$, either the edge or one of its endpoint nodes u or v has failed. The problem is similar because we also consider probabilistic failures in the graph, however in our case we consider more global dependencies, as we look at all ancestors of a node. There are many other

hard sequencing and scheduling problems [2], but their objective is different as they typically aim at minimizing makespan and do not model computer failures.

A probabilistic model similar to our model is studied by Sarmenta [22] but for independent tasks. Tasks are generated in batches. A batch consists of n tasks. There are p workers. Once a batch is generated, its tasks are assigned to workers; recomputation is allowed, and redundant assignment is allowed, too. At most a fraction f of workers is faulty. A faulty worker returns incorrect results with probability s, independently of other results. The goal is to compute results of each task so that each result is "credible" enough (several measures of credibility are proposed). Author considers two basic mechanisms: (1) spot check a worker by verifying if the result computed is correct—this helps estimate worker reliability and exclude faulty workers from computation, thus reducing the fraction of faulty workers over time, (2) redundantly compute a task until a certain number of results agree—this helps increase confidence in a result despite possibility of workers being faulty. Author shows that the combination of the two techniques is advantageous. Results are validated using a simulation.

There are scheduling problems that arise in Internet Supercomputing, other than the problem studied in this paper. The papers of Rosenberg [19] and Rosenberg and Yurkewych [18] introduce a formalism for studying the problem of scheduling tasks so as to render tasks eligible for allocation to workers (hence for execution) at the maximum possible rate. This allows one to utilize workers well, and also lessen the likelihood of the "gridlock" that ensues when a computation stalls for lack of eligible tasks. The papers identify optimal schedules for several significant families of structurally uniform dags. The paper of Malewicz et al. [13] extends this work via a methodology for devising optimal schedules for a much broader class of complex dags. These dags are obtained via composition from a prespecified collection of simple building-block dags. The paper introduces a suite of algorithms that decompose a given dag to expose its building-blocks, and a priority relation on building-blocks. When the building-blocks are appropriately interrelated, the dag can be scheduled optimally. Motivated by the demonstration in [13] that certain dags cannot be scheduled optimally, Malewicz and Rosenberg [14] formulate a scheduling paradigm in which tasks are allocated to workers in *batches* periodically. Optimality is always possible within this new framework, but achieving it may entail a prohibitively complex computation. However, restricted versions can be solved optimally in polynomial time. Malewicz et al. [15] show how to increase the speed of computation in the presence of network failures, by appropriately sequencing computation of disconnected workers.

Malewicz [12] introduces a parallel scheduling problem where a directed acyclic graph modeling t tasks and their dependencies needs to be executed on n unreliable workers. Worker i executes task j correctly with probability $p_{i,j}$. The goal is to find a regimen Σ, that dictates how workers get assigned to tasks (possibly in parallel and redundantly) throughout execution, so as to minimize the expected completion time. This fundamental parallel scheduling problem is shown to be NP-hard when restricted to constant dag width and also NP-hard

when restricted to a constant number of workers. These complexity results are contrasted with a polynomial time algorithm for the problem when both dag width and the number of workers are at most a constant.

6 Conclusions and Future Work

This paper began developing a scheduling theory for maximizing the expected number of correct results of tasks executed on unreliable computers, when tasks have dependencies. We introduced a combinatorial optimization problem, showed that the problem is NP-hard, and gave optimal polynomial time algorithms for restricted versions of the problem.

Our study opens several avenues for follow-up research. Which dags admit polynomial time optimal scheduling algorithms? Is there a constant factor approximation algorithm for the general problem? How to effectively schedule when each computer i has its own reliability p_i? How does possible asynchrony, or partial synchrony, (when tasks may take various amount of time to compute) affect scheduling decisions? One could consider a different optimization goal of maximizing the expected number of correctly computed sinks (in the case of one sink, we are then maximizing the likelihood that the sink will be correctly computed). Unreliability of computers could be modeled in a different way than probabilistically. We could assume that at most a certain number f of tasks will be incorrectly executed. We decide which tasks should be executed on a reliable computer, while an adversary decides which other at most f tasks will be executed incorrectly. Which tasks should be executed on a reliable computer, so as to maximize the worst-case (i.e., the lowest) number of correct results of tasks?

References

1. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (Im)possibility of Obfuscating Programs. (CRYPTO) (2001) 1–18
2. Crescenzi, P., Kann, V. (eds.): A compendium of NP optimization problems. `http://www.nada.kth.se/~viggo/wwwcompendium/node173.html`
3. Du, W., Jia, J., Mangal, M., Murugesan, M.: Uncheatable Grid Computing. 24th International Conference on Distributed Computing Systems (ICDCS) (2004)
4. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure, 2nd Edition. Morgan Kaufmann (2004)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)
6. Golle, P., Mironov, I.: Uncheatable Distributed Computations. RSA Conference – topics in Cryptography (2001) 425–440
7. Golle, P., Stubblebine, S.: Secure Distributed Computing in a Commercial Environment. 5th International Conference Financial Cryptography (FC) (2001) 289–304
8. The Intel Philanthropic Peer-to-Peer program. `http://www.intel.com/cure`
9. Kahney, L.: Cheaters Bow to Peer Pressure. Wired News, February 15 (2001) `http://www.wired.com/news/technology/0,1282,41838,00.html`

10. Kondo, D., Casanova, H., Wing, E., Berman, F.: Models and Scheduling Mechanisms for Global Computing Applications. 16th IEEE International Parallel & Distributed Processing Symposium (2002)
11. Korpela, E., Werthimer, D., Anderson, D., Cobb, J., Lebofsky, M.: SETI@home - massively distributed computing for seti. Computing in Science & Enginering, Vol. 3(1) (2001) 78–83
12. Malewicz, G.: Parallel Scheduling of Complex Dags under Uncertainty. (2005) submitted for publication
13. Malewicz, G., Rosenberg, A.L., Yurkewych, M.: On Scheduling Complex Dags for Internet-Based Computing. 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS) (2005) to appear
14. Malewicz, G., Rosenberg, A.L.: On batch-scheduling dags for Internet-based computing. Typescript, University of Massachusetts (2004) submitted for publication
15. Malewicz, G., Russell, A., Shvartsman, A.: Distributed Cooperation During the Absence of Communication. 14th International Symposium on Distributed Computing (DISC) (2000) 119–133
16. The Olson Laboratory Fight AIDS@Home project. http://www.fightaidsathome.org
17. Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Applied Mathematics, Vol. 131(3) (2003) 651–654
18. Rosenberg, A.L., Yurkewych, M.: Optimal Schedules for Some Common Computation-Dags on the Internet. IEEE Transactions on Computers (2005) to appear
19. Rosenberg, A.L.: On Scheduling Mesh-Structured Computations on the Internet. IEEE Transactions on Computers, Vol. 53(9) (2004)
20. Rosenberg, A.L.: Accountable Web-computing. IEEE Transactions on Parallel and Distributed Systems, Vol. 14(2) (2003) 97–106
21. The RSA Factoring By Web project. `http://www.npac.syr.edu/factoring`
22. Sarmenta, L.F.G.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems, Vol. 18(4) (2002) 561–572
23. SETI@home: Current Total Statistics. `http://setiathome.ssl.berkeley.edu/totals.html`, May 9 (2004)
24. Sun, X.H., Wu, M.: GHS: A performance Prediction and Task Scheduling System for Grid Computing. 17th IEEE International Parallel & Distributed Processing Symposium (2003)
25. Szajda, D., Lawson, B., Owen, J.: Hardening Functions for Large Scale Distributed Computations. IEEE Symposium on Security and Privacy, (2003) 216–224