

On the NP-Hardness of the REDUNDANTTASKALLOC Problem *

Joshua Wingstrom

Henri Casanova

Information and Computer Sciences Dept.
University of Hawai'i at Manoa, Honolulu, U.S.A.

Abstract

Consider an application that consists of n independent identical tasks to be executed on m computers, with $m > n$. Assume that each computer can execute a task with a given probability of success (typically < 1). One can use redundancy to execute replicas of some of the tasks on the $m - n$ computers. The problem is to determine how many replicas should be created for each task, or more precisely the number of task instances that should be created for each task and to which computers these instances should be allocated in order to maximize the probability of successful application completion. We formally define this problem, which we call REDUNDANTTASKALLOC, and prove that it is NP-hard.

1 Introduction

Consider a parallel application that requires that n identical and independent tasks $T_i, i = 1, \dots, n$, be completed. The target platform consists of m computers $P_j, j = 1, \dots, m$, with $m > n$. Each computer P_j can complete the computation of a task with probability $1 - p_j$ (p_j is the probability of task failure). Multiple instances of each task can be created and started in parallel on multiple computers with the goal of using redundancy to maximize the probability that the application completes successfully. Let us define $\delta_{i,j}$ to be 1 if an instance of T_i is assigned to P_j , and 0 otherwise. To enforce that a computer is assigned at most one task, we impose the constraint that for all $j = 1, \dots, m$, $\sum_{i=1}^n \delta_{i,j} = 1$. The probability that

*This paper is based upon work supported by the National Science Foundation under award number 0546688.

T_i completes successfully (i.e., that at least one of its instances completes successfully) is:

$$\prod_{j|\delta_{i,j}=1} (1 - p_j).$$

The probability that the entire application succeeds is then equal to:

$$\prod_i \left(1 - \prod_{j|\delta_{i,j}=1} p_j \right).$$

The problem is to find values for the $\delta_{i,j}$'s so that the above quantity is maximized. In this report we prove that this problem is NP-hard. Section 2 defines a special case of this problem, for $n = 2$, and proves its NP-hardness.

2 Problem Definition and NP-Hardness Proof

Consider the task allocation problem described in the previous section for the case of two tasks ($n = 2$). The problem becomes: for each computer, determine whether it should execute an instance of T_1 or an instance of T_2 . The problem can then be formalized as an optimization problem:

Definition 1. REDUNDANTTASKALLOC: Consider a set of m positive integers $S = \{a_1, \dots, a_m\}$, and $D = \text{lcm}(2^{a_1}, \dots, 2^{a_m})$. Find a subset A of S such that

$$\left(1 - \prod_{a_i \in A} \frac{2^{a_i}}{D} \right) \left(1 - \prod_{a_i \notin A} \frac{2^{a_i}}{D} \right)$$

is maximum.

Note that in the above formulation, and without loss of generality, we defined the failure probability p_j as a rational number between 0 and 1 of the form $2^q/D$.

An expression of the form $(1 - x)(1 - y)$, with the product $z = xy$ fixed, is maximized when $x = y$. Indeed, replace y by z/x , derivate with respect to x and find that the maximum is achieved for $x = y = \sqrt{z}$. This easily generalizes to stating that $(1 - x)(1 - y)$ is minimized when $|x - y|$ is minimized. Therefore, problem REDUNDANTTASKALLOC can be reformulated as:

Definition 2. REDUNDANTTASKALLOC: Consider a set of m positive integers $S = \{a_1, \dots, a_m\}$, and $D = \text{lcm}(2^{a_1}, \dots, 2^{a_m})$. Find a subset A of S such that

$$\left| \prod_{a_i \in A} \frac{2^{a_i}}{D} - \prod_{a_i \notin A} \frac{2^{a_i}}{D} \right|$$

is minimal.

Given that \log_2 is a strictly increasing function, an equivalent formulation is:

Definition 3. REDUNDANTTASKALLOC: Consider a set of m positive integers $S = \{a_1, \dots, a_m\}$. Find a subset A of S such that

$$\left| \sum_{a_i \in A} a_i - \sum_{a_i \notin A} a_i \right|$$

is minimal.

We consider the associated decision problem:

Definition 4. REDUNDANTTASKALLOC (K): Consider a set of m positive integers $S = \{a_1, \dots, a_m\}$. Is there a subset A of S such that

$$\left| \sum_{a_i \in A} a_i - \sum_{a_i \notin A} a_i \right| \leq K?$$

The reader will note that the above problem is very closely related to the well-known PARTITION decision problem [1]:

Definition 5. PARTITION: Consider a set of m positive integers $S = \{a_1, \dots, a_m\}$. Is it possible to find a subset A of S such that

$$\sum_{a_i \in A} a_i = \sum_{a_i \notin A} a_i?$$

If REDUNDANTTASKALLOC (K) is solvable in polynomial time, then so is PARTITION. Indeed, just solve REDUNDANTTASKALLOC (K) for $K = 0$ and answer "yes" to PARTITION is the answer to REDUNDANTTASKALLOC (K) is "yes", and "no" otherwise. However, the PARTITION problem is known to be (weakly) NP-complete. Therefore, REDUNDANTTASKALLOC (K) is also NP-complete, and it follows that REDUNDANTTASKALLOC is NP-hard. The proof is complete.

References

- [1] Michael R. Garey and David S. Johnson, *Computer and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, 1979.