Discrete Mathematics and Theoretical Computer Science DMTCS vol. **14:1**, 2012, 55–68

Optimal Computer Crash Performance Precaution

Efraim Laksman† Håkan Lennerstad‡ Lars Lundberg§

Blekinge Institute of Technology, Sweden

received 18th April 2011, revised 19th December 2011, accepted 18th January 2012.

For a parallel computer system with m identical computers, we study optimal performance precaution for one possible computer crash. We want to calculate the cost of crash precaution in the case of no crash.

We thus define a tolerance level r meaning that we only tolerate that the completion time of a parallel program after a crash is at most a factor $r + 1$ larger than if we use optimal allocation on $m - 1$ computers. This is an r-dependent restriction of the set of allocations of a program. Then, what is the worst-case ratio of the optimal r-dependent completion time in the case of no crash and the unrestricted optimal completion time of the same parallel program? We denote the maximal ratio of completion times $f(r, m)$ - *i.e.*, the ratio for worst-case programs. In the paper we establish upper and lower bounds of the worst-case cost function $f(r, m)$ and characterize worst-case programs.

Keywords: parallel computer, scheduling, computer crash, load balancing, process allocation, optimization

1 Introduction

Availability and performance are important in computer systems, and clusters of computers are often used to obtain both Pfister (1998). In order to obtain high performance we want to distribute the work load as evenly as possible between the computers in the cluster. High availability is obtained by redistributing the workload when computers break down. Most high availability systems are designed to handle one failure, *i.e.*, the system should redistribute the load from a crashed computer to the other computers. Multiple failures are extremely unlikely and the designers of high availability systems are in most cases satisfied if their system can continue to operate in case one computer goes down.

When a computer goes down, the processes on the crashed computer must be redistributed to the other computers. One would like to avoid excessive redistribution of processes. Therefore, we would like to move only the processes on the crashed computer, *i.e.*, the processes on the other computers should not be redistributed.

In Graham (1969) a tight upper bound is established for w_L/w_0, where w_L is optimal allocation with a restricted allocation set, and w_0 is the similar quantity with unrestricted allocation. The allocation restriction is in terms of a certain allocation algorithm. A similar but improved result, due to an improved

†Email: `efl@bth.se`
‡Email: `hln@bth.se`
§Email: `llu@bth.se`

algorithm, is given in Coffman Jr. et al. (1978). Finally, a polynomial time approximation scheme to multiprocessor scheduling has been presented by Hochbaum and Shmoys (1987). In the present paper, however, the restriction is not given by an algorithm, but by the specified crash deterioration tolerance. The results are independent of specific algorithms, so they are valid for all algorithms.

There has been a lot of research on performance of multiprocessors involving processor failures. In Girault et al. (2009) multiprocessor performance is optimized by a new method according to completing time and realiability of schedule when computer failures occur under certain restrictions. Ever et al. (2009) study the performance of Beowulf clusters under the assumptions of exponential failure and repair rates. The paper Al-Rousan and Shaout (2004) investigates the performance of large scale multiprocessors assuming a Weibull failure process. In contrast, the present paper assumes no statistical distribution for failures, nor any specific multiprocessor architecture. Furthermore, the performance issue is turned around in the sense that the accepted tolerance to performance loss in case of a failure is the starting point, and the result is the performance in case of no failure. This formulation of the problem appears so far to be rather unique in the field.

Note that the approach taken here differs from reoptimization. In reoptimization, the solution to a problem is solved more efficiently, in some respect, by exploiting the solution of a similar problem. This is not the case in the present research, since here the result consists in comparing two optimal solutions - how much deterioration that may occur in the case of no crash if certain well specified precautions are taken if expecting a computer crash. The computation of one of them is not taken advantage of to calculate the other. The present results do not concern how to calculate the bounds, in fact they involve NP-hard problems (see Section 5).

Many (real-time) systems have strict performance requirements that must be met also when one computer is down. The critical aspect in such systems is to keep the load on the most heavily loaded computer under a certain threshold. Since we only redistribute the processes on the crashed computer, the performance when one computer is down may depend on the identity of the crashed computer. For instance, consider a simple case with three identical computers and four processes: process one and two have a performance requirement of two performance units (*e.g.* MIPS) each, and processes three and four have a performance requirement of one unit each. Process one executes on computer one, and process two executes on computer two, whereas processes three and four share computer three. This means that under normal conditions when all computers are up and running the load is evenly balanced, *e.g.* the load is two on each computer. We will refer to the case when all computers are up and running as the *normal-case scenario*. If computer one or two goes down the load on some computer will be four, but if computer three goes down it is possible to put process three on computer one and process four on computer two and in that case the load on the most heavily loaded computer is only three. We will refer to the case when the most unfavorable computer is down as the *worst-case scenario*.

The optimal normal-case and optimal worst-case allocations are often not compatible, *i.e.*, one can in most cases not obtain optimal worst-case performance and optimal normal performance based on the same allocation of processes to computers. In the case above with three computers and four processes, one can obtain optimal worst-case performance by allocating processes one and three to computer one and processes two and four to computer two and leave computer three empty. If computer one or two goes down, the load on the crashed computer is simply moved to computer three. However, using that allocation the normal-case performance will be the same as the worst-case performance. If we spread out the load as evenly as possible we obviously get better normal-case performance, but as we discussed above the worst-case performance is lower since the load on the most heavily loaded computer could be as high as four.

After a computer break down we assume that all processes on that computer are reallocated optimally on the other computers. We also assume the worst possible break down, *i.e.*, by the computer that maximizes the resulting completion time after reallocation.

In this paper a tolerance level parameter r is introduced. We tolerate that the worst-case completion time after a computer crash is within a factor $1+r$ from the optimal completion time on $m-1$ computers – *i.e.*, when we allocate optimally assuming that one computer break down. Only allocations that fulfill this condition are considered. This is r-restricted allocation – we optimize the completion time of a program in the normal case within this restricted set of allocations. Note that $0 \leq r \leq 1$ where $r = 0$ means that we allow no deterioration after a crash, no tolerance, and $r = 1$ represents no restriction on the set of allocations.

We compare two normal-case completion times: r-restricted completion time to non-restricted. For a parallel program P, the function $f(P, r, m) \geq 1$ denotes the ratio of these two quantities. If A denotes an allocation, A_P the program P allocated by A, and $T(A_P)$ the completion time of A_P, we thus define

$$f(P, r, m) = \frac{\min_{A \ r\text{-restricted}} T(A_P)}{\min_A T(A_P)} \ .$$

Furthermore, we consider worst-case programs, *i.e.*, programs P maximizing $f(P, r, m)$:

$$\hat{f}(r, m) = \max_P f(P, r, m) \ .$$

In this paper we present upper and lower bounds on $\hat{f}(r, m)$. The upper bound of $\hat{f}(r, m)$ is deduced by identifying a certain subset of worst-case programs, and taking advantage of the properties of these programs. The lower bound is deduced by an entirely different method. It is derived from the properties of certain specific programs. The bounds meet for some values of m and r.

A related problem has previously been studied in Lundberg and Svahnberg (2003). In that case, the authors defined an algorithm for allocating processes to computers so that the performance is at most a factor 1.5 of the optimal performance for the normal-case scenario while at the same time guaranteeing that the performance when one computer has crashed is withing a factor of 1.5 of the optimal performance of the worst-case scenario. This means that from the existence of this algorithm we know that $\hat{f}(0.5, m) \leq 1.5$.

The problem of optimizing the performance of the worst-case scenario when more than one computer crash has been studied by Klonowska et al. for the special case when there is only one process on each computer, and this process requires one performance unit Klonowska et al. (2005). In this scenario the reallocations of processes on a crashed computer is done on one specific running computer according to a global recovery scheme. This scheme provides maximum load balancing if it is designed according to a so called Golomb ruler. It turns out that the scheme can be further improved by means of a modular Golomb ruler Klonowska et al. (2005).

Reliability in multiprocessor systems has been studied previously, *e.g.* in Chen and Cherkassky (1992). In that paper a static schedule based on rotations of an optimal schedule for the normal case is used, and the probability that all processes are run at least once is given as a function of the number of processors, number of rotations used and probability of break down for a single processor. Note the differences between Chen and Cherkassky (1992) and the present paper: here we assume the existence of at most one computer breakdown, allow reallocation of processes from a computer which has broken down, consider optimal schedules, which may be hard to find, rather than a specific schedule, and examine completion time, rather than the probability of complete break down.

2 Mathematical formulation of the problem

By the term *normal case* we mean no crash and no restriction on the set of allocations for a program. We start by defining notation.

- m is the number of identical computers in the cluster.
- P is a parallel program, *i.e.*, a set of independent processes.
- A_P is an allocation of P, *i.e.*, some way of distributing the processes in P among m computers.
- $T_i(A_P)$ is the completion time of the i:th computer of program P with allocation A.
- $T(A_P)$ is the time required to run A_P, *i.e.*, $T(A_P) = \max_{i=1,\ldots,m} T_i(A_P)$.
- $W(P)$ is the total work required to complete P, *i.e.*, $W(P) = \sum_{i=1}^{m} T_i(A_P)$.
- Φ_P denotes an allocation of P which is optimal in the normal case, hence $T(\Phi_P) \leq T(A_P)$ for any allocation A.

The notation chosen is not standard, but it is easy to read, avoiding an overflow of subscripts and superscripts, even when considering workload on different computers for different allocations of different programs at the same time.

We next describe the assumptions on a computer crash. Having a program P allocated with A_P, we assume that one of the m computers crash. The first assumption is that the processes on the computer that goes down are distributed on the remaining computers in an optimal manner: to minimize global execution time. Furthermore, there are m different computers that may crash. The second assumption is that the computer that crashes is the one that increases the completion time the most. A modification of A_P from m to $m-1$ computers according to these assumptions is denoted by A'_P.

We have thus defined the mapping $A_P \to A'_P$. In the allocation A'_P two assumptions are built in: the crash is worst possible, and it is a best possible reallocation of A_P.

We thus have two optimality concepts: one with a crash and one without, which requires two more definitions:

- A'_P is the best reallocation of A_P given the worst-case scenario.
- Ψ_P is an allocation of P where Ψ'_P is optimal: $T(\Psi'_P) \leq T(A'_P)$ for any allocation A.

There may be several allocations of P that are optimal in the normal case. If it is so, then Φ_P should be chosen among these allocations as the one which results in $T(\Phi'_P)$ being minimal.

Similarly, there may be several allocations of P that are optimal in the worst case. If it is so, then Ψ_P should be chosen among these allocations as the one which results in $T(\Psi_P)$ being minimal.

Using these definitions, the r-restriction on an allocation A_P is the following condition on A'_P

$$T(A'_P) \leq T(\Psi'_P)(1 + r) .$$

Within this set of allocations, we are interested in one that gives minimal normal-case completion time $T(A_P)$, *i.e.*, an allocation A where

$$\min_A \{T(A_P) : T(A'_P) \leq T(\Psi'_P)(1+r)\} .$$

The function f is the ratio to the normal-case completion time, so we define

$$f(P, r, m) = \frac{\min_A \{T(A_P) : T(A'_P) \leq T(\Psi'_P)(1+r)\}}{T(\Phi_P)} .$$

We seek worst-case behaviour of f over all programs P

$$\hat{f}(r, m) = \max_P f(P, r, m) .$$

It is immediate from the problem formulation that $\hat{f}(r, m)$ is non-increasing in the parameter r.

We next define teminology that will be used in this paper.

Definition 1 *A computer in the computer cluster is* critical *for the allocation A if it finishes its execution at $T(A_P)$. An allocation is called a* box allocation *(or just "box") if all computers are critical. A program P is called a* worst program *for r and m if $f(P, r, m) = \hat{f}(r, m)$. An allocation A_P for a program P is called* undominated *if and only if there is no allocation B_P with*

$$\begin{aligned} T(B_P) &\leq T(A_P) \textit{ and } T(B'_P) < T(A'_P), \textit{ or} && (1) \\ T(B_P) &< T(A_P) \textit{ and } T(B'_P) \leq T(A'_P) . && (2) \end{aligned}$$

An allocation which is not undominated is called dominated.

For all r and P there is an undominated allocation A_P that minimizes

$$\{T(A_P) : T(A'_P) \leq T(\Psi'_P)(1+r)\} . \tag{3}$$

To show this, assume that a dominated allocation B_P minimizes (3) for some r. Since B_P is dominated, there exists an undominated allocation A_P with $T(A_P) \leq T(B_P)$ and $T(A'_P) \leq T(B'_P)$. Obviously also A_P minimizes (3).

If there are two undominated allocations A and B with $T(A_P) = T(B_P)$ and $T(A'_P) = T(B'_P)$, we disregard from one of them. It follows now from the definition of undominated allocations that we have a certain set of undominated allocations

$$\{\Phi_P, A_{P,1}, A_{P,2}, \ldots, A_{P,n-2}, \Psi_P\}$$

with only strict inequalities as follows

$$\begin{aligned} T(\Phi_P) &< T(A_{P,1}) < \ldots < T(A_{P,n-2}) < T(\Psi_P) \leq \\ &\leq T(\Psi'_P) < T(A'_{P,n-2}) < \ldots < T(A'_{P,1}) < T(\Phi'_P), \end{aligned} \tag{4}$$

except possibly in $T(\Psi_P) \leq T(\Psi'_P)$. This gives the following characterization of the function f:

$$f(P,r,m) = \begin{cases} \frac{T(\Phi_P)}{T(\Phi_P)} = 1 & , \quad \frac{T(\Phi'_P)}{T(\Psi'_P)} \leq 1+r \\ \frac{T(A_{P,1})}{T(\Phi_P)} & , \quad \frac{T(A'_{P,1})}{T(\Psi'_P)} \leq 1+r < \frac{T(\Phi'_P)}{T(\Psi'_P)} \\ \frac{T(A_{P,2})}{T(\Phi_P)} & , \quad \frac{T(A'_{P,2})}{T(\Psi'_P)} \leq 1+r < \frac{T(A'_{P,1})}{T(\Psi'_P)} \\ \dots & , \quad \dots \\ \frac{T(\Psi_P)}{T(\Phi_P)} & , \quad 1+r < \frac{T(A'_{P,n-2})}{T(\Psi'_P)} \end{cases}$$

Note also that $T(\Phi'_P) \leq 2T(\Phi_P)$ is a trivial relationship, which in a way limits how much "unequal" the inequlities in (4) can be.

3 Upper bound on $\hat{f}$

We start by establishing some basic properties which are true for at least some worst program P for each pair of r and m.

Lemma 1 *For every r and m, there is a worst program P such that Φ_P or Ψ'_P is a box.*

Proof: Assume that there is a program P such that neither Φ_P nor Ψ'_P is a box. This means that there is at least one computer in each of the two allocations which is not critical. We choose a non-critical computer in Φ_P and call it a_0 and a non-critical computer in Ψ'_P and call it a_1. We now create the program Q with the same processes as P and an extra process α with $T(\alpha) = \min(T(\Phi_P) - T(a_0), T(\Psi'_P) - T(a_1))$. We have that $T(\Phi_P) = T(\Phi_Q)$ and $T(\Psi'_P) = T(\Psi'_Q)$. We also have that for any allocation A_Q, one can form an allocation A_P with $T(A_P) \leq T(A_Q)$ and $T(A'_P) \leq T(A'_Q)$, by allocating all processes from P to the same computer as they are allocated in A_Q. It is now clear from the definition of f that $f(P,r,m) \leq f(Q,r,m)$, so that if P is a worst program, then so is Q.

If there are several non-critical computers to start with, we may have to repeat the argument several times. Eventually we get a worst program for which Φ_P or Ψ'_P is a box. The lemma is proved. □

Lemma 2 *If P is a worst program for $r < \frac{m-2}{m}$ and Ψ'_P is a box, then also Φ_P is a box.*

Proof: Assume that Ψ'_P is a box and that Φ_P is not a box. We prove that then P cannot be a worst program since a specific program Q has higher value of f.

By (4), we always have $f(P,r,m) \leq \frac{T(\Psi_P)}{T(\Phi_P)}$. We will have $T(\Psi_P) \leq T(\Psi'_P) = \frac{W(P)}{m-1}$, as Ψ'_P is a box. We will also have $T(\Phi_P) > \frac{W(P)}{m}$, as Φ_P is not a box.

Now we have that

$$f(P,r,m) \leq \frac{T(\Psi_P)}{T(\Phi_P)} < \frac{W(P)/(m-1)}{W(P)/m} = \frac{m}{m-1}.$$

We next study the program Q with $m-1$ processes of length 1 and $m-1$ processes of length $m-1$. Then $T(\Phi_Q) = m-1$ and $T(\Phi'_Q) = 2(m-1)$, and $T(\Psi'_Q) = T(\Psi_Q) = m$. Since $T(A_Q) \geq m$ for any other allocation A, it is clear that Φ_Q and Ψ_Q are the only undominated allocations of Q. The condition $T(A'_P) \leq T(\Psi'_P)(1+r)$ with $A = \Phi$ now gives the r-values where Φ is allowed. We get

$$f(Q,r,m) = \begin{cases} \frac{m}{m-1} & , \quad r < \frac{m-2}{m} \\ 1 & , \quad r \geq \frac{m-2}{m} \end{cases} .$$

Thus P cannot be a worst program for $r < \frac{m-2}{m}$, since for these values $f(P,r,m) < f(Q,r,m)$. $\Box$

Corollary 3 *If $r < \frac{m-2}{m}$, there is a worst program P such that Φ_P is a box.*

Proof: This follows directly from Lemma 1 and 2. $\Box$

Lemma 4 *If P is a worst program for $r \geq \frac{m-2}{m}$ and Φ_P is a box, then $f(P,r,m) = 1$.*

Proof: Assume that $T(\Phi_P) = 1$. We know that Φ_P is a box allocated on m computers, so $W(P) = m$. We know also that Ψ'_P is allocated on $m-1$ computers only, so by $W(P) = m$ we have $T(\Psi'_P) \geq \frac{m}{m-1}$.

For any program P, if $\frac{T(\Phi'_P)}{T(\Psi'_P)} \leq 1 + r$ the allocation Φ_P is allowed so $f(P,r,m) = 1$. We get the implication $f(P,r,m) > 1 \Rightarrow 1 + r < \frac{T(\Phi'_P)}{T(\Psi'_P)}$. Using $T(\Phi'_P) \leq 2T(\Phi_P) = 2$ and $T(\Psi'_P) \geq \frac{m}{m-1}$ now gives us

$$f(P,r,m) > 1 \Rightarrow 1 + r < \frac{T(\Phi'_P)}{T(\Psi'_P)} \leq \frac{2(m-1)}{m} \ .$$

We can also express the implication as $f(P,r,m) > 1 \Rightarrow r < \frac{m-2}{m}$, which is equivalent to the statement of the lemma. $\Box$

Corollary 5 *If $r \geq \frac{m-2}{m}$ and $f(P,r,m) > 1$, there is a worst program P such that Ψ'_P is a box.*

Proof: This follows directly from Lemma 1 and 4, and from that $f(P,r,m) \geq 1$ for all P, r and m where it is defined. $\Box$

We next present the upper bounds of $\hat{f}$ in two theorems.

Theorem 6 *For any m, we have that*

$$\hat{f}(r,m) < \frac{2}{1+r}, \text{ for } 0 \leq r < 1 \ .$$

Proof: For any program P, we have

$$\hat{f}(r,m) \leq \frac{T(\Psi_P)}{T(\Phi_P)} \leq \frac{T(\Psi'_P)}{T(\Phi_P)} \tag{5}$$

$$\frac{1}{1+r} > \frac{T(\Psi'_P)}{T(\Phi'_P)} \tag{6}$$

$$2T(\Phi_P) \geq T(\Phi'_P) \ . \tag{7}$$

The inequalities (5) and (7) are trivial. The inequality (6) follows from that we would have $f(P,r,m) = 1$ otherwise.

Joining these inequalities we get

$$\hat{f}(r,m) \leq \frac{T(\Psi'_P)}{T(\Phi_P)} \leq 2\frac{T(\Psi'_P)}{T(\Phi'_P)} < \frac{2}{1+r} \ .$$

$\square$

From the proof of this theorem, we see that if a worst program is to meet the given bound, it must contain at least one process of length $T(\Phi_P)$ and Φ_P must be a box. But for $r \geq \frac{m-2}{m}$ we know that Φ_P is not a box, so we can get a stronger bound in this case.

Theorem 7 *For $\frac{m-2}{m} \leq r \leq \frac{m-2}{m-1}$, we have $\hat{f}(r,m) \leq \frac{m-2}{r(m-1)}$, and for $\frac{m-2}{m-1} < r$, we have $\hat{f}(r,m) = 1$.*

Proof: Since $r \geq \frac{m-2}{m}$, we have that for a worst program P, Φ_P is not a box. This means that in Φ_P, there will be a computer which runs for the minimum amount of time, $x < T(\Phi_P)$. We immediately get $T(\Phi'_P) \leq T(\Phi_P) + x$, as a crash can cause nothing worse than placing the load of a critical computer on the computer with the lightest load. From this we conclude that $T(\Phi_P) + x \geq T(\Phi'_P) \geq (1+r)T(\Psi'_P)$, as we would otherwise have $\hat{f}(r,m) = 1$, which is a trivial lower bound.

Since Φ_P has one computer with run time $T(\Phi_P)$, one computer with run time x and the other computers have run time of at least x and at most 1, we get $(m-1)T(\Phi_P) + x \geq W(P) \geq T(\Phi_P) + (m-1)x$. We will only need the lower bound on $W(P)$. On the other hand, we have that Ψ'_P is a box. Distributing P equally on all $(m-1)$ computers, we get

$$T(\Psi'_P) = \frac{W(P)}{m-1} \geq \frac{T(\Phi_P)}{m-1} + x \ .$$

We get

$$T(\Phi_P) + x \geq (1+r)T(\Psi'_P) \geq (1+r)\left(\frac{T(\Phi_P)}{m-1} + x\right) \ .$$

Solving for x now gives

$$\frac{(m-2-r)T(\Phi_P)}{r(m-1)} \geq x \ .$$

But we have

$$f(P,r,m) \leq \frac{T(\Psi'_P)}{T(\Phi_P)} \leq \frac{1}{T(\Phi_P)} \frac{T(\Phi_P) + x}{1+r}$$

Replacing with our bound on x, we get rid of $T(\Phi_P)$, and get

$$f(P,r,m) \leq \frac{1 + \frac{m-2-r}{r(m-1)}}{1+r} = \frac{rm + m - 2 - 2r}{r(1+r)(m-1)} = \frac{m-2}{r(m-1)} \ .$$

However, we know that $\hat{f}$ is trivially bounded by 1 from below, so that the above reasoning cannot hold for $r > \frac{m-1}{m-2}$, where Φ_P must be permitted. $\square$

4 Lower bound on $\hat{f}$

The worst programs we have found so far provide our lower bounds. We remark that not all of our examples here result in either Φ or Ψ being a box. The programs can be altered, according to the proof of Lemma 1, in such a way that Φ or Ψ becomes a box, but this will only complicate calculation of the value of f (by introducing additional undominated allocations), without changing the value of f in the region (in terms of r and m) where the given program gives an interestingly high value of f.

While Lemma 8 and Corollary 9 have not been explicitly used to find any bound on $\hat{f}$, they have been helpful when searching for worst programs, and may be helpful to anyone intending to continue this line of research.

Lemma 8 *For any r and m, there is a worst program P which does not have any pair of processes on the same computer in both allocations Φ_P and Ψ'_P.*

Proof: Assume that there is a program P which has a pair of processes on the same computer in both allocations Φ_P and Ψ'_P, such that $f(P, r, m) = \hat{f}(r, m)$. Then, consider the program Q which is identical to P, except that the two processes allocated to the same computer in both Φ_P and Ψ'_P are merged into one process of length equal to the sum of the two previous length.

We have that $T(\Phi_P) = T(\Phi_Q)$ and $T(\Psi'_P) = T(\Psi'_Q)$. We also have that for any allocation A_Q, one can form an allocation A_P with $T(A_P) = T(A_Q)$ and $T(A'_P) \leq T(A'_Q)$ (by allocating processes in A_P just as they are allocated in A_Q). It is now clear from the definition of f that $f(P, r, m) \leq f(Q, r, m)$, so that if P is a worst program, then so is Q.

If P carried large groups processes on the same computer in both Φ_P and Ψ'_P, then the argument above may have to be repeated several times, but eventually we will have a worst program with the sought after condition. $\Box$

Corollary 9 *For every r and every m, there is a worst program P, such that every computer in Φ_P executes at most $m - 1$ processes and every computer in Ψ'_P executes at most m processes.*

Proof: By the pigeon hole principle, if a computer in Φ_P executes more than $m - 1$ processes, then at least 2 of these processes must wind up on the same computer in Ψ'_P, in which case the program is uninteresting according to the lemma above.

Similarly, if a computer in Ψ'_P executes more than m processes, then at least 2 of these processes must come from the same computer in Φ_P, in which case the program is uninteresting according to the lemma above. $\Box$

4.1 First program

Consider the program P with $k \geq 2$ processes of length 1 and $m - 1$ processes of length k, when we have $m \geq k+1$ computers. It is fairly easy to see that $T(\Phi_P) = k$, $T(\Phi'_P) = 2k$ and $T(\Psi_P) = T(\Psi'_P) = k+1$. See Figure 1.

Clearly no other allocations can be undominated, as they would require at least as much time as Ψ_P, even before a crash. Thus we get

$$f(P, r, m) = \frac{k+1}{k}, \text{ when } 1 + r < \frac{2k}{k+1} .$$

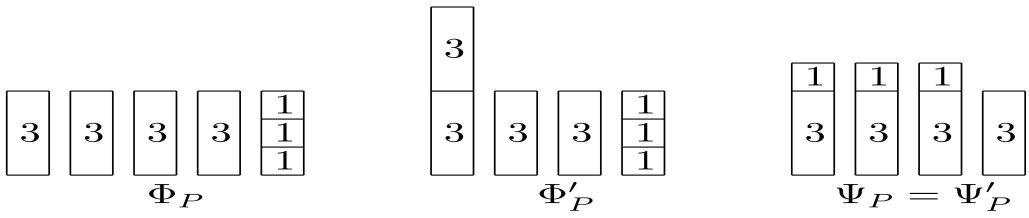

Fig. 1: Example with $m = 5,\ k = 3$.

4.2 Second program

Consider the program P with $k-1$ processes of length 1 (we want $k \geq 2$), one process of length $k-1+\epsilon$, with $0 \leq \epsilon < 1$ and $m-2$ processes of length $k-1$, when we have $m \geq k+1$ computers. It is fairly easy to see that $T(\Phi_P) = k-1+\epsilon$, $T(\Phi'_P) = 2(k-1)+\epsilon$ and $T(\Psi_P) = T(\Psi'_P) = k$. See Figure 2.

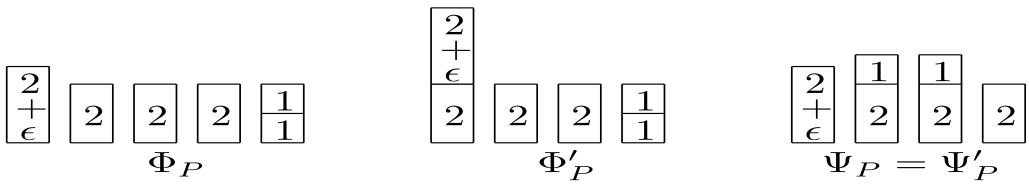

Fig. 2: Example with $m = 5,\ k = 3$.

Clearly no other allocations can be undominated, as they would require at least as much time as Ψ_P, even before a crash. Thus we get

$$f(P, r, m) = \frac{k}{k-1+\epsilon}, \text{ when } 1 + r < \frac{2(k-1)+\epsilon}{k} \;.$$

"Solving" the condition for ϵ, one attains $\epsilon > kr - k + 2$. We now set $\epsilon = kr - k + 2 + \epsilon'$, with $\epsilon' > 0$. This gives us

$$f(P, r, m) = \frac{k}{kr+1+\epsilon'} \;.$$

Now we let ϵ' approach 0, so that $f(P, r, m)$ gets arbitrarily close to $\frac{k}{kr+1}$.

4.3 Combining the programs

Since we have $\hat{f}(r, m) \geq f(P, r, m)$ for all programs P, we have

$$\hat{f}(r) \geq \max\left(\frac{k+1}{k}, \frac{k}{kr+1} - \epsilon\right), \text{ for } k \text{ such that } \frac{k-2}{k} \leq r < \frac{k-1}{k+1},$$

for arbitrarily small $\epsilon > 0$. If we calculate which term the maximum function will "choose" for which r, we may express it as

$$\hat{f}(r,m) \geq \begin{cases} \frac{k}{kr+1} - \epsilon, & \frac{k-2}{k} \leq r \leq \frac{k-1}{k+1} - \frac{1}{k^2+k} \\ \frac{k+1}{k}, & \frac{k-1}{k+1} - \frac{1}{k^2+k} \leq r \leq \frac{k-1}{k+1} \end{cases}$$

given that m is large enough, which it may be considered to be when

$$r < \frac{m-2}{m} \Leftrightarrow m > \frac{2}{1-r},$$

because this allows usage of the programs and allocations described above. In the case where $m \leq \frac{2}{1-r}$, we may still use the second kind of program described, with $k = m$. This gives the following lower bound. For arbitrarily small ϵ we have

$$\hat{f}(r,m) \geq \frac{m}{mr+1} - \epsilon, \text{ if } \frac{m-2}{m} \leq r < \frac{m-1}{m}.$$

We gather these bounds in a theorem:

Theorem 10 *We have*

$$\hat{f}(r,m) \geq \begin{cases} \frac{k}{kr+1} - \epsilon, & \frac{k-2}{k} \leq r \leq \frac{k-1}{k+1} - \frac{1}{k^2+k} \\ \frac{k+1}{k}, & \frac{k-1}{k+1} - \frac{1}{k^2+k} \leq r \leq \frac{k-1}{k+1} \end{cases}, \textit{ for } r \in \left[0, \frac{m-2}{m}\right),$$

$$\hat{f}(r,m) \geq \frac{m}{mr+1} - \epsilon, \textit{ for } r \in \left[\frac{m-2}{m}, \frac{m-1}{m}\right)$$

and

$$\hat{f}(r,m) \geq 1, \textit{ for } r \in \left[\frac{m-1}{m}, 1\right],$$

for any $\epsilon > 0$.

5 Conclusions

When allocating processes to computers in a cluster there is a trade-off between normal-case and worst-case performance. In this paper we have been able to quantify this trade-off. The function $\hat{f}(r,m)$ makes it possible to determine how close to the optimal normal-case performance we can get, given that we accept a worst-case performance within a factor of $1 + r$ of optimal worst-case performance.

For instance, consider a cluster of at least 4 computers (*i.e.*, $m \geq 4$) and a scenario where we accept worst-case performance of 1.5 times the optimal worst-case performance (*i.e.*, $r = 0.5$). In that situation, Figure 3 shows that for every program it is possible to meet this restriction and obtain a normal-case performance of at most $1.33\ldots$ times the optimal normal-case performance. However, finding the optimal allocation A for given parameters P, r, m is NP-hard. This can be seen as in the case $r = 1$, finding A is equivalent to finding Φ which contains the NP-complete problem of multiprocessor scheduling (on m computers), while in the case $r < 1$, one must determine $T(\Psi'_P)$, which also contains the problem of multiprocessor scheduling (on $m - 1$ computers). For proof of the NP-completeness, see *e.g.* Garey and Johnsson (1979).

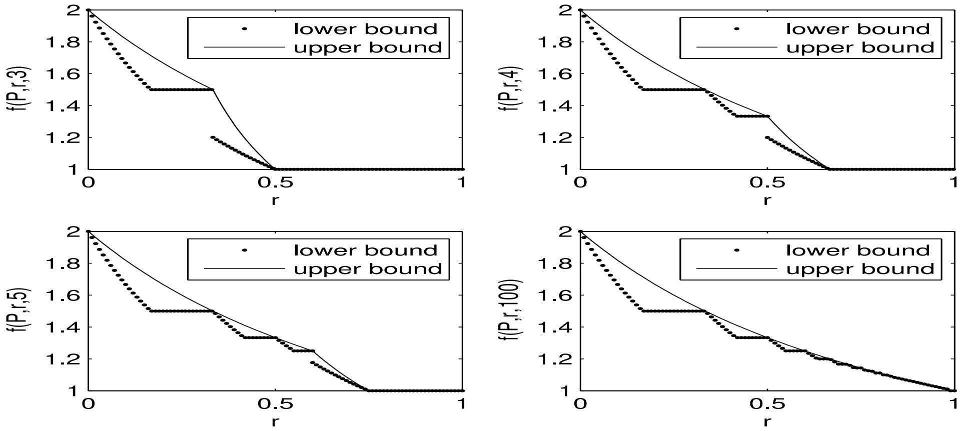

Fig. 3: Upper and lower bounds for different numbers of computers.

Also reallocating optimally, *i.e.*, finding the best A'_P for a given A_P, is NP-hard. This can be seen by considering the case with 3 computers. No matter which computer crashes, reallocating the processes originally allocated to it corresponds to finding a 2-partition of the union of those processes and a process of length equal to the difference between the original load on the two other computers, a problem which is known to be NP-hard Garey and Johnsson (1979).

As can be seen in Figures 3 and 5 there is very little space between the upper and lower bound. The slack of the bounds is greatest in the proximity of $r = \frac{1}{6}$ for all m and $r = \frac{1}{3}$ at $m = 3$. An open question is how the bounds can be improved.

6 Acknowledgements

The authors would like to thank reviewer B for comments regarding computational complexity and intractability and thank Prof. Bengt Aspvall for discussions on the same topic.

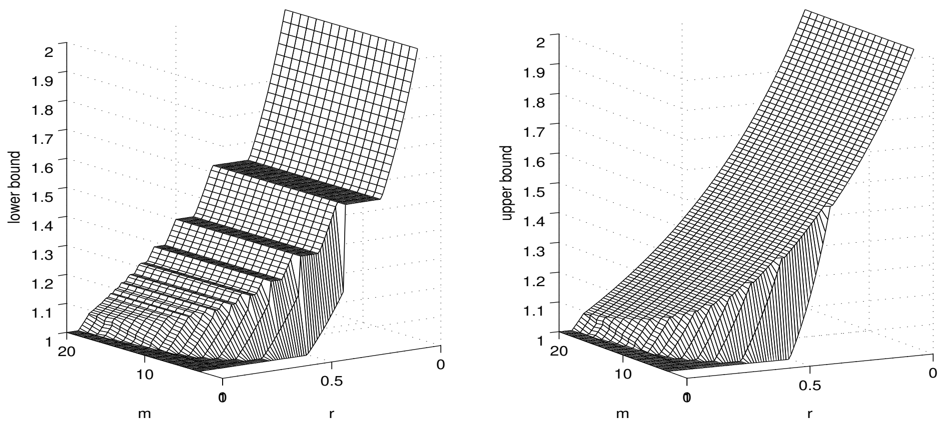

Fig. 4: Lower and upper bound of $\hat{f}$.

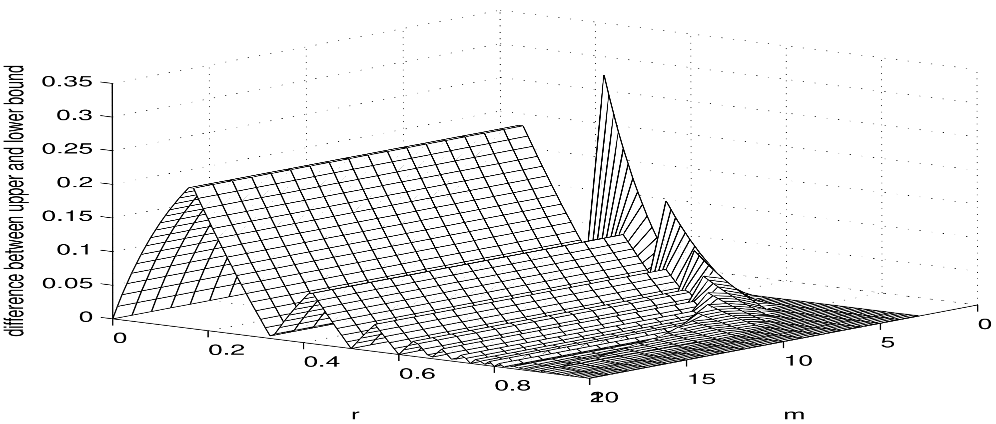

Fig. 5: The difference between the upper and lower bound of $\hat{f}$.

References

M. Al-Rousan and A. Shaout. Closed-form solution for reliability of sci-based multiprocessor systems using weibull distribution and self-healing rings. *Computers and Electrical Engineering*, 30:309–329, 2004.

C.-I. H. Chen and V. Cherkassky. Redundant task-allocation in multicomputer systems. *IEEE Transactions on Reliability*, 41, 1992.

E. Coffman Jr., M. Garey, and D. Johnsson. An application of bin packing to multiprocessor scheduling. *SIAM Journal of Computing*, 7(1):1–17, 1978.

E. Ever, O. Gemikonakli, and R. Chakka. Analytical modelling and simulation of small scale, typical and highly available beowulf clusters with breakdowns and repairs. *Simulation Modelling Practice and Theory*, 17:327–347, 2009.

M. Garey and D. Johnsson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN:978-0716710455.

A. Girault, É. Saule, and D. Trystram. Reliability versus performance for critical applications. *Journal of Parallel and Distributed Computing*, 69:326–336, 2009.

R. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2): 416–429, 1969.

D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the ACM*, 34, 1987.

K. Klonowska, H. Lennerstad, L. Lundberg, and C. Svahnberg. Optimal recovery schemes in fault tolerant distributed computing. *Acta Inf.*, 41(6):341–365, 2005.

L. Lundberg and C. Svahnberg. Normal versus worst-case performance in high-availability cluster and distributed computing. In *Proceedings of the 21st IASTED Multiconference on Applied Infromatics, Innsbruck, Austria*, 2003.

G. Pfister. *In search of clusters*. Prentice Hall, 1998. ISBN: 0-13-899709-8.