On the Cost of Task Re-Scheduling in Fault-Tolerant Task Parallel Computations

C. Bertolli  
Dept. Computer Science  
Univ. Pisa – Italy

J. Gabarró  
Dept. of Llenguatges i Sistemes Informatics  
Univ. Politecnica de Catalunya – Spain

CoreGRID Technical Report  
Number TR-TR-0114  
September 30st, 2007

Institute on Programming Model

CoreGRID - Network of Excellence  
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme  
Project no. FP6-004265
On the Cost of Task Re-Scheduling in Fault-Tolerant Task Parallel Computations

C. Bertolli
Dept. Computer Science
Univ. Pisa – Italy

J. Gabarró
Dept. of Llenguatges i Sistemes Informatics
Univ. Politecnica de Catalunya
– Spain

Abstract
Fault tolerance is still an “hot” topic in the context of Grid computing. There exist several solutions and techniques to face this issue, some of them inherited from the fault tolerance research, some other introduced in the context of Grid computing. Anyway, poor work has been done in providing such fault tolerance strategies in a lightweight, scalable and analyzable way. In this technical report we address the last aspect, by considering an abstract model of computation for a class of structured parallel programs (namely farm computations) to analyze the overhead incurred by parallel computations in the case of failures. We present the evolution of our study showing the model we exploit, and its properties, and the results we obtained. We also completely define some base cases, and we present experimental results to validate our approach.

1 Introduction
Fault tolerance (FT) is a central issue in the context of Grid computing platforms [8]. There exist several works introducing and studying fault tolerance strategies for such execution environments. Our approach is to study fault tolerance for structured parallel programs (e.g. skeletons) [7, 11, 3, 1, 2]. The models of structured parallelism feature properties that are known at compile-time, like the patterns of interaction between the parallel executors of a parallel computation. Our aim is to show that such properties can be used to: (a) introduce simple and optimized FT strategies, and (b) analyze their impact on the performance. In this document we focus on task parallel computations, and we study the overheads that a simple FT strategy induces on their performance in the case of faults.

There exists several strategies to introduce fault tolerance in these kind of computations [4, 5]. One of the most straightforward and lightweight is the task re-scheduling technique: in the case of failure of the execution of a task, it is re-scheduled to a free executor. This strategy is implemented in the support of the muskel library [3], that we choose as research tool for our tests. Task re-scheduling is a kind of cold task redundancy technique, in that a cold copy of the task is kept for successive re-scheduling. Differently, in hot task redundancy techniques, usually called task replication, the execution of a same task is performed in parallel on different resources.

The problem faced in this study consists in analyzing, from a mathematical viewpoint, the behavior of the execution of a given set of tasks on a given set of resources, in the presence of resource failure and restart. Each task has a known probability of success/failure. The goal is to obtain a static evaluation of the expected completion time of the computation, as an upper bound of the actual running time.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).
1.1 Programming and Computation Model

Our study on the run-time quantitative behavior of parallel computations is based on the muskel programming environment [3]. A muskel program is a macro data-flow graph composed of sequential and parallel modules, connected through stream (i.e. possibly unlimited sequences of typed elements). Parallel modules are expressed as skeletons, and can be nested in arbitrary hierarchies. Examples of skeletons that can be used to implement a parallel algorithm are farm, and pipeline. Farm computations express a task parallel computation, where a set of workers performs the same program on different input data (obtained from an input stream), and delivering an output stream of results (one for each task). In a pipeline computation, each element received from an input stream is passed to a set of nested functions (e.g. \( \text{out} = \text{F}_1(\text{F}_2(\ldots \text{F}_n(\text{in}) \ldots)) \)), and the results are delivered to an output stream. Each function \( \text{F}_i \) is implemented in a different stage, and the evaluation of two functions on different input elements can happen in parallel.

As example, in Fig. 1 we show the graph part of a muskel program that specifies the graph of the program. Two farms, defined at lines 1 and 3, are composed in a pipeline of two stages, defined at line 5. The implementation maps the whole main module in a single slave. The stream of input data, on which we have to perform the computation is scheduled in an on-demand strategy to slaves. The master, along with scheduling, is also responsible of collecting results from slaves.

The implementation of muskel is based on a master-slave strategy: the master schedules the input tasks to slaves. A whole muskel program, i.e. a data-flow graph, is mapped in each slave, which perform the computation locally, on different tasks, and independently w.r.t. each other.

We abstractly define the implementation model of muskel to allow its mathematical formulation. The model consists in a set of N tasks performed on M resources. Each task is performed independently with each other. We assume that each task can be performed in an average execution time, denoted with the symbol \( \delta \). In the most general case, the number of tasks could be not known in advance.

1.2 Failure Model and Fault Tolerance Strategies

1.2.1 Muskel FT Strategy

In this description, we will conversely use the terms slave and remote interpreter, mixing up the parallelism unit of muskel (the slave) and the process implementing it (the remote interpreter). The FT strategy is implemented in the master process, which structure is represented in Fig 2. We denote processes with squares, threads with circles, and state variables with rounded rectangles. The master includes two pools, one for tasks, and the other for their results. Each slave is managed by an independent thread (slave \( S_i \) is managed by \( CT_i \), where CT stands for Control Thread). Control threads and slaves are linked with bold double-ended arrows, denoting the task scheduling and result collection operations. Control threads are also linked to tasks and results to obtain and collect them (arrows half dotted, half dashed). We also exploit a restart detection thread (denoted with RD), that is responsible of receiving restart messages from slaves (dotted arrows). The behavior in the case of slave failure is represented in Fig. 3: slave \( S_1 \) fails (1) and the corresponding bold arrow with its control thread is closed (2). The corresponding control thread detects the closing of the connection, and it terminates (3). In the implementation, a list of active slaves is also exploited. In the event of failure, the control thread removes the slave from the list of actives. The list is used to control the degree of parallelism of the computation. The behavior in the case of slave restart is represented in Fig. 4. Slave \( S_1 \) is restarted by some external mechanism (1), and it communicates the RD thread on the master of its availability (2). RD spawns a new control thread for the restarted slaves (3).
1.2.2 Abstract Failure Model

We introduce an abstract failure model and two instances of it. From a general viewpoint, the first instance we present is more similar to the abstract model than the second one, which better simulates the behavior of an actual execution environment subject to failures. We characterize our failure models w.r.t. qualitative and quantitative aspects. Features of the first kind describe the behavior of a resource when it fails, and the class of the failure detection sub-systems [6]. Features of the second kind describe the frequency of faults, their patterns, and their duration. We characterize instances of the abstract model by implementing in different ways the quantitative parameters. The qualitative ones are the same for the abstract model and its instances.

The abstract failure model can be characterized by considering two orthogonal aspects:

**Qualitative** Computational resources fail by stopping their execution, according to the fail-stop model [9], i.e. after a failure a resource stops to perform its task. After a failure, a computational resource is eventually restarted. The detectability of failures and restarts are demanded to a sub-system.

**Quantitative** The failure of a resource is subject to probability q. Conversely, we denote with p (p = 1 − q) the probability of success of a resource. The failure is expressed in terms of slave invocations: every t invocations to a slave there is a q probability of failure. The time needed to detect a failure is represented by the variable $T_F$. After a failure, it takes a time $T_R$ for a resource to be restarted. No constraints are given for the value that $T_R$ can assume, the fault frequency and their patterns.

In both instances of the abstract model we can set the $p$ value. The way in which the failure and restart latencies are controlled characterize the two instances.

**First Instance of the Failure Model** In the first instance of the abstract failure model, failures can happen every time a slave is invoked, i.e. $t = 1$. The time needed to detect a failure can be upper bounded, but it cannot be directly controlled: the actual value can vary at each failure and, in general, it will depend on some lower virtualization level features. We denote with $\Delta_F$ the upper bound on failure detectability. The time needed to restart an interpreter is a random variable in the range $0 \leq T_R \leq \Delta_R$, and we can set the upper bound. Also, we can decide the probability distribution the restart time.

**Second Instance of the Failure Model** In the second instance of the abstract failure model, failures can happen every $t \geq 1$ time a slave is invoked. We can control both the time needed to detect a failure, and the time needed to restart it, by specifying their upper bounds and the failure distributions they follow. The constraints on the values of $T_F$ and $T_R$ are: $0 \leq T_F \leq \Delta_F$, and $0 \leq T_R \leq \Delta_R$. Probability distributions of $T_F$ and $T_R$ can be possibly different. In the experiments we can study different configurations of these parameters to address actual execution environments.

![Graphical representation of the implementation of the FT strategy of muskel](image)

Figure 2: Graphical representation of the implementation of the FT strategy of muskel.
1.2.3 Implementation of Fault Injection and Experimental Setup

The simulation avoids to restart resources as: (a) we can control, at software level, the restart latencies of processes to emulate actual fault detection and restart latencies for resources; (b) the failure and restart system is more scalable; (c) typically system management in a cluster (resource halt and restart) requires root permissions.

We implemented two versions of the fault injector system, one for each instance described above.

Implementation of the First Instance of the Failure Model  Failures of calls to a slave are implemented as part of their behavior (see Fig.5): when a request for task execution is received, the slave performs a part of the task for a random time. Next, a random number is generated and, depending if it is lower than a fixed value q, the slave either executes the tasks or terminates with failure. The q value can be configured on each slave as an initialization parameter. Eventually, we will allow modification of this value at run-time to simulate environments in which the failure probability of resources can change during the computation. Notice that the \( t = 1 \) feature is obtained by tossing a coin at each invocation, in the remote interpreter code.

The restart of a slave is performed by an external process that monitors for slave failures and, when it is the case, it restarts them. We will call this module the Restarter. A parameter for the monitoring is the time between two successive fault detections (this value should be chosen to minimize the overhead it causes to the application, and the restart latency). This value represents the upper bound over the fault detection latency, i.e. \( \Delta_F \).

In Fig. 6 we show the implementation of the first instance of the abstract failure model. Each Interpreter (denoted with \( I_i \)) is augmented with a fault injector (denoted with \( finj \)). The master process, running on a robust node, is...
RemoteInterpreter: :

... 

compute(Task t) {
    //compute for random time...
    //...
    double coin = rand();
    if(coin := p) –
        exit(FAILURE);
    ~
    //compute the remaining of the task
    Result r = t.execute();
    return r;
}

Figure 5: Fault injection code in the remote interpreters. At each invocation: (a) we perform a sub-part of the task for a random time; (b) we toss a coin and we decide to fail or terminate the execution; (c) we exit or we terminate the execution of the task, and we return the result.

The restarter subsystem is implemented as a single process and is executed on a robust node. Whenever a slave fails (1) it detects the failure in $T_F$ time (2), and it re-spawns it (3) in $T_R$ time (dotted arrows). In the figure we show the failure of $I_n$, and the re-spawning of a new instance of it from the restarter. After the restart the manager re-connects to the new instance of $I_n$.

Implementation of the Second Instance of the Failure Model We exploit a single process to terminate slaves (i.e. inject faults), and to restart them. Every $T_C$ seconds the program chooses if it has to terminate a slave (with probability $q$). If it is the case, it terminates the slave. Next, the program waits for $T_F$ seconds to simulate the failure probability and it waits $T_R$ seconds before restarting the slave.

$T_C$ represents the grain at which the failure can happen, and is a random variable, according to some probability distribution, upper bounded by the value $\Delta_C$ that is specified for each machine. We claim that this implementation is correct w.r.t. its specification, as, given that the average completion time of each task is $T_T$, the $t$ value can be obtained as: $t = \lceil \frac{T_T}{T_C} \rceil$. It is important to notice that, differently from the previous model, the decision to fail is taken independently of the frequency at which an interpreter is called, i.e. it is application independent.

As in the model, $T_F$ and $T_R$ can follow specific probability distributions. Differently from the previous model $T_F$ is not upper bounded by the monitoring frequency, but it is directly simulated and can be controlled to study different situations.

The input of the program specifies the names of the injected resources, and for each one of them the $\Delta_F$ and $\Delta_R$ values and the probability distributions for choosing $T_F$ and $T_R$. It outputs a log file describing each failure/restart event, and their timing, in absolute (absolute time) and w.r.t. the beginning of the injection (elapsed time).
Figure 6: Representation of the first instances of the fault injection technique. Whenever a slave fails (1) it detects the failure in $T_F$ time (2), and it re-spawns it (3) in $T_R$ time (dotted arrows).

2 Base Cases

In this section we show the models based on Markov chains for simple cases, and we study the expected average completion time for such cases, and the variance for one of them. We also show some experimental results for one of the cases.

Notation We have seen that the probability of success in executing a task is denoted with $p$, and that of failure with $q = 1 - p$. We denote with $\delta$ the average time needed to perform a task. We denote with $\Delta$ the time needed to: (a) perform part of the task (denoted with $\delta/2$, as we assume failure to be equally probable at all time of the task execution); (b) detect a failure (denoted with $\Delta_F$); (c) restart the slave (denoted with $\Delta_R$). Clearly, we address the restart of the slave on the same computational resource, or on another one added to the set of active ones. That is, the abstract model does not include the information on the mapping between processes and computational resources. Finally, we denote with $\mu$ the maximum between $\delta$ and $\Delta$, i.e. $\mu = \max\{\delta, \Delta\}$.

2.1 1 Task performed by 1 Interpreter

We study the most simple case of the behavior of the execution of a single task on a single resource, in the case of failure.

2.1.1 Model

In Fig. 7 we represent the Markov chain related to the execution of a single task on a single interpreter. In the case of failure, we re-schedule the task on the restarted interpreter, according to the FT strategy. In the figure, circles represent states of the Markov chain, and they are labeled with the number of tasks that have not still yet executed. Each transition, denoted with an arrow from a state to another state (possibly the same), is labeled with: (a) one of the symbols in $\{F,S\}$, denoting respectively the failure or success of execution, and (b) with the probability of taking the corresponding transition. In this example we start from the state labeled with 1, denoting that we have to execute a task, and:

- In the case of failure we transit again in the same state.
- In the case of success we transit in the state labeled with 0.

This last state represents the ending of the computation. We represented also an arc from the state 0 to itself labeled with probability 1 for modeling purposes. In fact, when all tasks are finished, we remain in the last state(s) we reached,
labeled with 0. In the next figures, we will avoid to represent the arc from a termination state to itself. We also avoid to show F,S symbols, as they should be clear from the values of the state labels. The graph of Fig. 7 is a Markov chain because:

- It has a finite number of states: the one labeled with 0 and the one with 1.
- Each arc is labeled with a probability value greater or equal to 0.
- For each state, the sum of the probability on its output arcs is equal to one.

We also claim that it is an absorbing Markov chain, because there is one absorbing state (the one labeled with 0), and from all the other states (i.e. just the one labeled with 1) the absorbing one can be reached in a finite number of steps (i.e. 1 step).

We will see that all the graphs that we derive from the examples are absorbing Markov chains, and we will exploit their properties to derive the expected number of re-tries for each task. In Fig. 8 we add timing information on arcs. We assume that $\Delta = \Delta_F + \Delta_R + (\delta/2)$ (see below) and that $\delta$ is the average time needed to perform a task. That is, in the case of failure of one of the interpreters, we have to wait for the detection of the fault and for the restart of the interpreter. Also, the failure happens some time after the execution has started, so we have to count at most the average completion time for a task in the $\Delta$ value.

To compute the average number of re-tries, given the probability $p$ of success, we can model each try of execution as a Bernoulli process, obtaining that the whole process is modeled as a geometric distribution. We are interested in the average number of failed re-tries before the first success, i.e. in the mean value for the geometric variable that is known to be $1/p$. We can compute the average time needed to execute the whole computation as:

$$\Delta p + (\Delta + \delta)qp + (2\Delta + \delta)q^2p + (3\Delta + \delta)q^3p + \ldots$$

The first term of the expression represents the case without failures. We multiply the time needed to perform the task with the probability of having no failures. The following terms represent increasing numbers of failures, and are
multiplied by the corresponding overheads (i.e. $\Delta$ time for each failure, and $\delta$ time for the last successful execution).

We can divide the expression in two terms by multiplying the sums:

$$\delta p + \delta q p + \delta q^2 p + \delta q^3 p + \ldots + \Delta q p + 2\Delta q^2 p + 3\Delta q^3 p + \ldots$$

and we can take a $\lambda p$ term and a $\Delta p$ term out of each sub-expression respectively:

$$\delta p \left(1 + q + q^2 + q^3 + \ldots\right) + \Delta p \left(q + 2q^2 + 3q^3 + \ldots\right)$$

We can reduce the two expressions between the parentheses to their sum. The first one is a geometric sum and it converges to $\frac{1}{1-q}$, i.e. $\frac{1}{p}$. The, the first term of the whole expression becomes:

$$\delta p \left(1 + q + q^2 + q^3 + \ldots\right) = \delta p \frac{1}{1-q} = \delta$$

The second expression between parentheses needs a derivation: if we define

$$u = q + q^2 + q^3 + q^4 + \ldots = \frac{1}{1-q}$$

and if we derive this expression we obtain:

$$\frac{du}{dq} = 1 + 2q + 3q^2 + 4q^3 + \ldots$$

Consequently we can derive the sum on the right and we obtain:

$$\frac{du}{dq} = 1 + 2q + 3q^2 + 4q^3 + \ldots = \frac{1}{(1-q)^2} = \frac{1}{p^2}$$

Thus, the second expression of the expected time can be reduced:

$$\Delta p \left(q + 2q^2 + 3q^3 + \ldots\right) = \Delta pq \left(1 + 2q + 3q^2 + 4q^3 + \ldots\right) = \Delta pq \frac{1}{p^2} = \Delta \frac{q}{p}$$

The expected time for the case of 1 task executed on 1 interpreter is:

$$\delta + \Delta \frac{q}{p}$$

### 2.2 2 Tasks performed by 2 Interpreters

In this subsection we show the Markov model of the execution of 2 tasks on 2 interpreters. We exploit the model to derive an average of the expected completion time, and its variance.

#### 2.2.1 Model

We consider the execution of 2 tasks on 2 interpreters (see Fig. 9) In the figure, the initial state is the one labeled with the value 2. Initially, we assign the two tasks to the two interpreters and:

- If both fail (edge labeled with $F,F$), we remain in the same state. This event has $q^2$ probability, and it takes $\Delta$ time.
- If one fails, and the other succeeds, we transit in the state labeled with 1 (edge labeled with $F,S$). This event has a probability equal to $2pq$, and it takes $\mu = \max\{\delta, \Delta\}$ to execute it. We use the max over the two times because, in the model, we have to wait for a results from both tasks to decide the next step.
- If both succeed, we transit to the state labeled with 0 (edge labeled with $S,S$). This event has a probability of $p^2$ and it takes $\delta$ time (i.e. the parallel execution time).

From the step labeled with 1 we re-assign the failed task to an interpreter (the first available?) and:

- In the case of failure, we remain in 1. This event has probability $q$ and it takes $\Delta$ time (i.e. we have to wait for failure detection and process restart for re-scheduling the task).
- In the case of success we transit in 0. This event has a probability $p$ and it takes $\delta$ time.

Both states labeled with 0 are absorbing.
2.2.2 Average Expected Time

We compute the average completion time by considering the two paths to the absorbing states independently. The computation is split in two sums:

\[ T_{comp} \leq \sum_{w \in Path_1} p(w) \cdot t(w) + \sum_{w \in Path_2} p(w) \cdot t(w) \]

where \( p(w) \) is the probability of following the path \( w \) and \( t(w) \) is the time needed for the path \( w \). \( Path_1 \) is the one that goes from 2 to 0 passing by the state labeled with 1. \( Path_2 \) is the one directly connecting 2 to 0, i.e. the one without failures.

We can further decompose the first longest path in two successive sub-paths:

- The first half goes from the state 2 to the state 1.
- The second half from state 1 to state 0.

Next we can compose the probability and timing of each sub-path. The timing information is simply additive: \( t(w_1, w_2) = t(w_1) + t(w_2) \), i.e. the time needed to perform the first and the second sub-paths is equal to the sum of the single sub-times. Also the probability of the two paths can be composed of the two single sub-probabilities, as shown in Fig. 10. We re-schedule \( k \) times both tasks until one of them succeeds. This is done with probability \( q^{2k} \cdot 2pq \). Next we re-schedule the remaining task \( j \) times until it succeeds. This is done with probability \( q^j \cdot p \). Because of the sequential behavior of the process, we have that the sub-probabilities are independent: \( p(w_1, w_2) = p(w_1) \cdot p(w_2) \).

We can decompose the whole expression of the expected time by considering that probabilities are independent, and time values feature additivity:

\[ t(w_1, w_2) \cdot p(w_1, w_2) = [t(w_1) + t(w_2)] \cdot p(w_1) \cdot p(w_2) \]

Applying it to the first path (from 2 to 0 passing by 1) we have that:

\[ \sum_{(w_1, w_2)} t(w_1, w_2) \cdot p(w_1, w_2) = \sum_{(w_1, w_2)} [t(w_1) + t(w_2)] \cdot p(w_1) \cdot p(w_2) \]

By multiplying the two times for the probabilities we have:

\[ \sum_{(w_1, w_2)} t(w_1) \cdot p(w_1) \cdot p(w_2) + \sum_{(w_1, w_2)} t(w_2) \cdot p(w_1) \cdot p(w_2) \]
Figure 10: Scheme of the behavior of retries for the longest path, in the case of one failure and one success. In this behavior we first iterate on the state 2 (with 2 failures), next we obtain a success and a failure and we transit in the state 1. The state 1 is iterated until a success is obtained.

If we multiply the single independent probabilities of the two sub-paths we obtain:

$$
\sum_{w_1=2 \rightarrow 1} t(w_1) \cdot p(w_1) \cdot \sum_{w_2=1 \rightarrow 0} p(w_2) + \sum_{w_1=2 \rightarrow 1} p(w_1) \cdot \sum_{w_2=1 \rightarrow 0} t(w_2) \cdot p(w_2)
$$

The two sums for single probabilities can be solved:

$$
\sum_{w_2=1 \rightarrow 0} p(w_2) = 1
$$

and

$$
\sum_{w_1=2 \rightarrow 1} p(w_1) = \frac{2pq}{1-q^2}
$$

The whole upper bound on the expected time is:

$$
T_{comp} \leq \sum_{w_1=2 \rightarrow 1} t(w_1) \cdot p(w_1) + \frac{2pq}{1-q^2} \cdot \sum_{w_2=1 \rightarrow 0} t(w_2) \cdot p(w_2) + \sum_{w_3=2 \rightarrow 0} t(w_3) \cdot p(w_3)
$$

In the next subsections we consider the three quantities independently.

**First Sub-Expression** We resolve the expression:

$$
\sum_{w_1=2 \rightarrow 1} t(w_1) \cdot p(w_1) = \mu \cdot 2pq + (\Delta + \mu)q^2 \cdot 2pq + (2\Delta + \mu)q^4 \cdot 2pq + (3\Delta + \mu)q^6 \cdot 2pq + \ldots
$$

We perform the multiplication of $\Delta$ and $\mu$ to obtain two sums. Next we take out $2pq$ and $2\mu pq$ respectively from each sum and we define $u = q^2$:

$$
2\mu pq(1 + u + u^2 + u^3 + \ldots) + 2pq(\Delta u + 2\Delta u^2 + 3\Delta u^3 + \ldots).
$$

The first sum converges to $\frac{2p_2}{1-q^2}$ while the second one:

$$
\Delta u + 2\Delta u^2 + 3\Delta u^3 + \ldots = u(\Delta + 2\Delta u + 3\Delta u^2 + \cdots) = u\Delta(1 + 2u + 3u^2 + \ldots) = \ldots
$$
The expression between parentheses is the derivation of a geometric sum, thus we can reduce the whole expression to:

\[ \ldots = u \cdot \frac{\Delta}{(1 - u)^2} = \Delta \frac{q^2}{(1 - q^2)^2} \]

The whole first sub-expression becomes:

\[ \sum_{w_2=1} t(w_2) \cdot p(w_2) = 2\mu \frac{pq}{1 - q^2} + 2pq\Delta \frac{q^2}{(1 - q^2)^2} \]

**Second Sub-Expression** We resolve the expression:

\[ \sum_{w_2=1} t(w_2) \cdot p(w_2) \]

Recall from the case of 1 task executed on 1 interpreter that

\[ T_{comp} \leq \delta + \Delta \frac{q}{p} \]

The second sub-expression becomes:

\[ \sum_{w_2=1} t(w_2) \cdot p(w_2) = \frac{2pq}{1 - q^2} \left[ \delta + \Delta \frac{q}{p} \right] \]

**Third Sub-Expression** We resolve the expression:

\[ \sum_{w_3=2} t(w_3) \cdot p(w_3) \]

We can express it as:

\[ \delta p^2 + (\Delta + \delta)q^2p^2 + (2\Delta + \delta)q^4p^2 + (3\Delta + \delta)q^6p^2 + \cdots \]

We multiply the factors between parentheses and we obtain two sums

\[ \delta p^2(1 + q^2 + q^4 + q^6 + \cdots) + p^2\Delta(q^2 + 2q^4 + 3q^6 + \cdots) \]

The first sum converges to \( \frac{1}{1-q^2} \). If we define \( u = q^2 \) the second one converges to:

\[ q^2(1 + 2q^2 + 3q^4 + \cdots) + q^2(1 + 2u + 3u^2 + \cdots) = q^2 \frac{1}{(1-u)^2} = \frac{q^2}{(1-q^2)^2} \]

The whole sub-expression becomes:

\[ \delta p^2 \frac{1}{1 - q^2} + p^2\Delta \frac{q^2}{(1 - q^2)^2} \]

**Summing Up the Three Parts** If we substitute the formulas we obtained for the three single sub-expressions we have that:

\[ T_{comp} \leq \mu 2pq \cdot \frac{1}{1 - q^2} + \Delta 2pq \cdot \frac{q^2}{(1 - q^2)^2} + \frac{2pq}{1 - q^2} \cdots \left[ \delta + \Delta \frac{q}{p} \right] + \delta p^2 \frac{1}{1 - q^2} + \Delta p^2 \cdot \frac{q^2}{(1 - q^2)^2} \]

We can reduce the second and fifth terms in the following way:

\[ \Delta 2pq \cdot \frac{q^2}{(1 - q^2)^2} + \Delta p^2 \cdot \frac{q^2}{(1 - q^2)^2} = \Delta \frac{q^2}{(1 - q^2)^2} = \Delta \frac{q^2}{(1 - q^2)^2} \cdot (1 - q^2) = \Delta \frac{q^2}{1 - q^2} \]
By algebraic simplifications:

\[ T_{\text{comp}} \leq \frac{2pq}{1-q^2} \left[ \delta + \mu + \Delta \frac{q}{1-q} \right] + \Delta \cdot \frac{q^2}{1-q^2} + \delta \cdot \frac{p^2}{1-q^2} = \frac{2pq}{1-q^2} (\delta + \mu) + \frac{2pq}{1-q^2} \Delta \cdot \frac{q}{p} + \Delta \cdot \frac{q^2}{1-q^2} + \delta \cdot \frac{p^2}{1-q^2} \]

Finally:

\[ T_{\text{comp}} \leq \frac{2pq}{1-q^2} (\delta + \mu) + 3\Delta \cdot \frac{q^2}{1-q^2} + \delta \cdot \frac{p^2}{1-q^2} \]

Below, we verify the correctness of this formula and the precision of this formula by experimenting the muskel fault-tolerant implementation.

2.2.3 Variance of the Model

We compute the variance value for the completion time. Considering the Markov model of the computation, we can express it as two paths (see Fig. 9):

Path 1 We iterate on state 2 (2 failures each iteration) \(k_1\) times, next we transit on state 1 (1 failure and 1 success), we iterate on the state 1 (1 failure) \(k_2\) times, and we finally transit in state 0 (1 success).

Path 2 We iterate on state 2 (as above, 2 failures each iteration) \(k_3\) times, and we directly terminate in state 0 (2 successes).

Clearly, \(k_1, k_2,\) and \(k_3\) span from 1 to \(\infty\). We consider the time and the probability of each path, and we compute the variance as:

\[ \text{Var}[X] = E[X^2] - E^2[X] \]

It is worth of notice that we exploit the path-based description to easily compute the \(E[X^2]\) term. We can exploit the expression of the expected time previously computed to obtain the second term (i.e. \(E^2[X]\)).

**Computation of \(E[X^2]\)** We have to compute \(E(\tau^2(w)) = \sum_w \tau^2(w)p(w)\). We split as

\[ \sum_w \tau^2(w)p(w) = \sum_{w:2\rightarrow1\rightarrow0} \tau^2(w)p(w) + \sum_{w:2\rightarrow0} \tau^2(w)p(w) \]

and we study the two paths independently. In the computations, we will exploit the following simple results:

\[ \sum_k q^k = \frac{1}{1-q} \]
\[ \sum_k kq^k = \frac{1}{(1-q)^2} \]
\[ \sum_k q^k = \frac{q+q^4}{(1-q)^3} \]

The probability and time of the first path follow:

\[ \tau(w) = \tau(k_1, k_2) = \Delta k_1 + \mu + \Delta k_2 + \delta \]
\[ p(w) = p(k_1, k_2) = (q^2)^{k_1} 2pq^{k_2} p \]

We develop as follows:

\[
\begin{align*}
((\Delta k_1 + \mu + \Delta k_2 + \delta)\cdot p(w) &= ((\Delta k_1 + \Delta k_2) + (\mu + \delta))\cdot p(w) \\
&= [((\Delta k_1 + \Delta k_2) + (\mu + \delta)\cdot (\mu + \delta)^2)] \cdot p(w) \\
&= [\Delta^2 k_1^2 + 2\Delta^2 k_1 k_2 + \Delta^2 k_2^2 + 2\Delta k_1 (\mu + \delta) + 2\Delta k_2 (\mu + \delta) + (\mu + \delta)^2] \cdot p(w) \\
&= \Delta^2 k_1^2 + 2\Delta^2 k_1 k_2 + \Delta^2 k_2^2 + 2\Delta k_1 (\mu + \delta) + 2\Delta k_2 (\mu + \delta) + (\mu + \delta)^2 \cdot p(w)
\end{align*}
\]
We consider each term independently and we obtain:

\[
A = \sum_{k_1, k_2} \Delta^2 k_1^2 \cdot (q^2)^{k_1} 2pqk_2^p = 2\Delta^2 pq \cdot \frac{q^2 + q^4}{(1 - q^2)^3}
\]

\[
B = \sum_{k_1, k_2} 2\Delta^2 k_1 k_2 (q^2)^{k_1} 2pqk_2^p = 4\Delta^2 \cdot \frac{q}{(1 - q^2)^2}
\]

\[
C = \sum_{k_1, k_2} \Delta^2 k_2^2 (q^2)^{k_1} 2pqk_2^p = 2\Delta^2 \cdot \frac{q^2}{p^2}
\]

\[
D = \sum_{k_1, k_2} 2\Delta k_1 (\mu + \delta)(q^2)^{k_1} 2pqk_2^p = 4\Delta (\mu + \delta) \cdot \frac{1}{p(1 + q)^2}
\]

\[
E = \sum_{k_1, k_2} 2\Delta k_2 (\mu + \delta)(q^2)^{k_1} 2pqk_2^p = 4\Delta (\mu + \delta) \cdot \frac{q}{1 - q^2}
\]

\[
F = \sum_{k_1, k_2} (\mu + \delta)^2(q^2)^{k_1} 2pqk_2^p = (\mu + \delta)^2 \cdot \frac{q}{1 + q}
\]

Clearly, the first path can be computed as:

\[
\sum_{2 \rightarrow 1 \rightarrow 0} \tau^2(w)p(w) = A + B + C + D + E + F
\]

The probability and time of the second path are:

\[
\tau(w) = \tau(k_3) = \Delta k_3 + \delta
\]

\[
p(w) = p(k_3) = (q^2)^{k_3} p^2
\]

We develop as follows:

\[
\sum_{k_3} (\Delta k_3 + \delta)^2 \cdot (q^2)^{k_3} p^2 = \sum_{k_3} (\Delta^2 k_3^2 + 2\Delta \delta k_3 + \delta^2) \cdot (q^2)^{k_3} p^2
\]

We develop each term independently, and we obtain:

\[
G = \sum_{k_3} \Delta^2 k_3^2 (q^2)^{k_3} p^2 = \Delta^2 p^2 \sum_{k_3} k_3^2 (q^2)^{k_3} = \Delta^2 \cdot \frac{q^2 + q^4}{p(1 + q)^3}
\]

\[
H = \sum_{k_3} 2\Delta \delta k_3 (q^2)^{k_3} p^2 = 2\Delta \delta p^2 \sum_{k_3} k_2(q^2)^{k_3} = 2\Delta \delta \cdot \frac{q^2}{(1 + q)^2}
\]

\[
I = \sum_{k_3} \delta^2 (q^2)^{k_3} p^2 = \delta^2 \sum_{k_3} (q^2)^{k_3} p^2 = \delta^2 \cdot \frac{p}{1 + q}
\]

As above, the result for the second path can be computed as:

\[
\sum_{2 \rightarrow 0} \tau^2(w)p(w) = G + H + I
\]

**Computation of** \(E^2[X]\) **We take the computed value for** \(E[X]\) **and we simply power it to 2:**

\[
E^2[X] = \left( \frac{2pq}{1 - q^2} (\delta + \mu) + 3\Delta \cdot \frac{q^2}{1 - q^2} + \delta \cdot \frac{p^2}{1 - q^2} \right)^2
\]
Variance Result  The complete formula for the variance is quite complex:

\[
Var[X] = E[X^2] - E[X]^2 = 
= \left( \sum_{w=1}^{n} \sigma^2(w)p(w) + \sum_{w=0}^{n} \sigma^2(w)p(w) \right) - \left( \frac{2pq}{1-q^2} (\delta + \mu) + 3\Delta \cdot \frac{q^2}{1-q^2} + \delta \cdot \frac{p^2}{1-q^2} \right)^2
= \left( A + B + C + D + E + FG + H + I \right) - \left( \frac{2pq}{1-q^2} (\delta + \mu) + 3\Delta \cdot \frac{q^2}{1-q^2} + \delta \cdot \frac{p^2}{1-q^2} \right)^2
\]

We avoid to show the complete unrolling of single sub-expressions for space reasons.

2.3 Tests for 2 tasks performed by 2 Interpreters

We have tested the case of 2 tasks performed by 2 interpreter in the muskel library. The results are characterized w.r.t. the value assume by \( \mu, \delta \) or \( \Delta \).

For each configuration, we compute the average value, and the variance, on the 100 runs we experimented. The average and variance values are computed as following:

\[
E = \frac{1}{n} \sum_{x=1}^{n} T_{\text{comp}}^x
\]

\[
Var = \frac{1}{n} \sum_{x=1}^{n} (T_{\text{comp}}^x - E)^2
\]

Where \( n = 100, T_{\text{comp}} \) is the set of the obtained completion times, and \( T_{\text{comp}}^i \) is the i-th result.

2.3.1 Experiments with \( \delta > \Delta \), i.e. \( \mu = \delta \)

In this subsection we consider configurations in which the time needed to detect a failure and restart the failed process is lower than the time needed to compute a task. In the implementation, in the case of failure, the task is rescheduled to the same interpreter that is restarted.

Failure Probability \( q = 0.1 \)  The configuration of these tests is: \( \delta = 10, \Delta = \Delta_F + \Delta_R = 1 + 4 = 5, \mu = \max\{\delta, \Delta\} = \delta = 10, p = 0.9, q = 0.1 \) If we apply the formula

\[
T_{\text{comp}} \leq 2(\delta + \mu) \cdot \frac{pq}{1-q^2} + 3\Delta \cdot \frac{q^2}{1-q^2} + \delta \cdot \frac{p^2}{1-q^2}
\]

we obtain an upper bound for the average completion time of the computation:

\[
T_{\text{comp}} \leq 2 \cdot (10 + 10) \cdot \frac{0.9 \cdot 0.1}{1 - (0.1)^2} + 3 \cdot 6 \cdot \frac{0.1^2}{1 - (0.1)^2} + 10 \cdot \frac{0.9^2}{1 - (0.1)^2}
\]

Resolving the equation:

\[
T_{\text{comp}} \leq 12
\]

Results  We have run 100 times the experiment and measured the completion time. Table 1 show the results of all tests and the average completion time. Notice that it is lower than the theoretical completion time computed in the previous paragraph. Some of the completion time we have monitored are greater than the theoretical one. In an actual environment the distribution of faults can induce such completion times, but it should be noticed we have monitored a low number of them. This is in some sense a consequence of the failure probability value.

The average completion time monitored (11.37679) is lower than the theoretical upper bound (12).
Table 1: Results of 100 tests of execution of 2 tasks on 2 interpreters, with $\delta = 10, \Delta = 5, p = 0.9$.

Failure Probability $q = 0.2$ The configuration parameters of these tests are: $\delta = 10, \Delta = \Delta_F + \Delta_R = 1 + 4 = 5, \mu = \max \delta, \Delta = \delta = 10, p = 0.8, q = 0.2$.

We apply the same formula of the previous subsection and we obtain:

$$T_{\text{comp}} \leq 13.95833$$

Results We have run 100 times the experiment and measured the completion time. Table 2 show the results of all tests and the average completion time. Notice that it is lower than the theoretical completion time computed in the previous paragraph. Again the average completion time monitored (12.83438) is lower than the theoretical upper bound (13.95833).

Failure Probability $q = 0.5$ The configuration of these tests is: $\delta = 10, \Delta = \Delta_F + \Delta_R = 1 + 4 = 5, \mu = \max \delta, \Delta = \delta = 10, p = 0.5, q = 0.5$.

We apply the same formula of the previous subsection and we obtain:

$$T_{\text{comp}} \leq 21.66667$$

Results We have run 100 times the experiment and measured the completion time. Table 3 show the results of all tests and the average completion time. Notice that it is lower than the theoretical completion time computed in the previous paragraph. Again the average completion time monitored (20.14205) is lower than the theoretical upper bound (21.66667).
Table 2: Results of 100 tests of execution of 2 tasks on 2 interpreters, with $\delta = 10$, $\Delta = 5$, $p = 0.8$.

2.4 Experiments with $\delta < \text{Delta}$, i.e. $\mu = \Delta$

In these tests we assume that the time needed to compute a task is lower than the time needed to detect a failure and restart the failed interpreter. In the case of failure, the corresponding lost task is scheduled to the non-failed interpreter.

**Failure Probability** $q = 0.1$ The configuration of these tests is: $\delta = 10$, $\Delta = \Delta_F + \Delta_R = 1 + 14 = 15$, $\mu = \max \delta, \Delta = \Delta = 15$, $p = 0.9$, $q = 0.1$.

We apply the formula:

$$T_{comp} \leq 2(\delta + \Delta) \frac{pq}{1-q^2} + 3\Delta \frac{q^2}{1-q^2} + \delta \frac{p^2}{1-q^2}$$

and we obtain:

$$T_{comp} \leq 13.18182$$

**Results** We have run 100 times the experiment and measured the completion time. Table 4 show the results of all tests and the average completion time. Notice that it is lower than the theoretical completion time computed in the previous paragraph. Again the average completion time monitored (13.07644) is lower than the theoretical upper bound (13.18182).

**Failure Probability** $q = 0.2$ The configuration of these tests is: $\delta = 10$, $\Delta = \Delta_F + \Delta_R = 1 + 14 = 15$, $\mu = \max \delta, \Delta = \Delta = 15$, $p = 0.8$, $q = 0.2$.

We apply the formula of the previous subsection and we obtain:

$$T_{comp} \leq 16.875$$
Table 3: Results of 100 tests of execution of 2 tasks on 2 interpreters, with $\delta = 10$, $\Delta = 5$, $p = 0.5$.

**Results**  We have run 100 times the experiment and measured the completion time. Table 5 show the results of all tests and the average completion time. Notice that it is lower than the theoretical completion time computed in the previous paragraph. Again the average completion time monitored (15.21683) is lower than the theoretical upper bound (16.875).

**Failure Probability** $q = 0.5$  The configuration of these tests is: $\delta = 10$, $\Delta = \Delta_F + \Delta_R = 1 + 14 = 15$, $\mu = \max{\delta, \Delta} = 15$, $p = 0.5$, $q = 0.5$.

We apply the formula of the previous subsection and we obtain:

$$T_{\text{comp}} \leq 35$$

**Results**  We have run 100 times the experiment and measured the completion time. Table 6 show the results of all tests and the average completion time. Notice that it is lower than the theoretical completion time computed in the previous paragraph. Again the average completion time monitored (26.57598) is lower than the theoretical upper bound (35).

### 2.4.1 Discrepancies between Experimental and Theoretical Results

In Table 7 we show a subsume of all values obtained in described tests and their corresponding theoretical values. Table 8 shows the same values characterized w.r.t the failure probabilities.
<table>
<thead>
<tr>
<th>Num.</th>
<th>$T_{\text{comp}}$</th>
<th>Num.</th>
<th>$T_{\text{comp}}$</th>
<th>Num.</th>
<th>$T_{\text{comp}}$</th>
<th>Num.</th>
<th>$T_{\text{comp}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10.6</td>
<td>2</td>
<td>10.342</td>
<td>3</td>
<td>10.376</td>
<td>4</td>
<td>21.917</td>
</tr>
<tr>
<td>5</td>
<td>10.409</td>
<td>6</td>
<td>10.366</td>
<td>7</td>
<td>10.355</td>
<td>8</td>
<td>23.448</td>
</tr>
<tr>
<td>13</td>
<td>23.448</td>
<td>14</td>
<td>10.4</td>
<td>15</td>
<td>10.365</td>
<td>16</td>
<td>10.366</td>
</tr>
<tr>
<td>17</td>
<td>10.365</td>
<td>18</td>
<td>10.367</td>
<td>19</td>
<td>20.392</td>
<td>20</td>
<td>20.908</td>
</tr>
<tr>
<td>25</td>
<td>10.291</td>
<td>26</td>
<td>18.859</td>
<td>27</td>
<td>17.836</td>
<td>28</td>
<td>10.424</td>
</tr>
<tr>
<td>29</td>
<td>10.346</td>
<td>30</td>
<td>10.381</td>
<td>31</td>
<td>10.348</td>
<td>32</td>
<td>10.372</td>
</tr>
<tr>
<td>33</td>
<td>10.366</td>
<td>34</td>
<td>10.29</td>
<td>35</td>
<td>10.359</td>
<td>36</td>
<td>10.414</td>
</tr>
<tr>
<td>37</td>
<td>21.399</td>
<td>38</td>
<td>10.375</td>
<td>39</td>
<td>10.356</td>
<td>40</td>
<td>22.444</td>
</tr>
<tr>
<td>41</td>
<td>10.334</td>
<td>42</td>
<td>10.349</td>
<td>43</td>
<td>10.372</td>
<td>44</td>
<td>20.9</td>
</tr>
<tr>
<td>45</td>
<td>20.395</td>
<td>46</td>
<td>10.371</td>
<td>47</td>
<td>10.35</td>
<td>48</td>
<td>19.862</td>
</tr>
<tr>
<td>49</td>
<td>10.375</td>
<td>50</td>
<td>22.431</td>
<td>51</td>
<td>10.468</td>
<td>52</td>
<td>10.368</td>
</tr>
<tr>
<td>57</td>
<td>10.386</td>
<td>58</td>
<td>15.799</td>
<td>59</td>
<td>22.433</td>
<td>60</td>
<td>10.384</td>
</tr>
<tr>
<td>61</td>
<td>10.382</td>
<td>62</td>
<td>10.364</td>
<td>63</td>
<td>10.367</td>
<td>64</td>
<td>10.354</td>
</tr>
<tr>
<td>69</td>
<td>10.357</td>
<td>70</td>
<td>24.462</td>
<td>71</td>
<td>23.96</td>
<td>72</td>
<td>25.055</td>
</tr>
<tr>
<td>73</td>
<td>17.841</td>
<td>74</td>
<td>10.402</td>
<td>75</td>
<td>10.359</td>
<td>76</td>
<td>10.385</td>
</tr>
<tr>
<td>77</td>
<td>10.362</td>
<td>78</td>
<td>20.408</td>
<td>79</td>
<td>10.362</td>
<td>80</td>
<td>10.358</td>
</tr>
<tr>
<td>81</td>
<td>10.376</td>
<td>82</td>
<td>10.353</td>
<td>83</td>
<td>10.369</td>
<td>84</td>
<td>10.376</td>
</tr>
<tr>
<td>85</td>
<td>10.355</td>
<td>86</td>
<td>10.357</td>
<td>87</td>
<td>10.341</td>
<td>88</td>
<td>10.367</td>
</tr>
<tr>
<td>89</td>
<td>24.473</td>
<td>90</td>
<td>10.4</td>
<td>91</td>
<td>10.358</td>
<td>92</td>
<td>10.383</td>
</tr>
<tr>
<td>93</td>
<td>10.365</td>
<td>94</td>
<td>10.366</td>
<td>95</td>
<td>10.367</td>
<td>96</td>
<td>20.386</td>
</tr>
<tr>
<td>97</td>
<td>10.384</td>
<td>98</td>
<td>10.358</td>
<td>99</td>
<td>18.342</td>
<td>100</td>
<td>21.919</td>
</tr>
</tbody>
</table>

Table 4: Results of 100 tests of execution of 2 tasks on 2 interpreters, with $\delta = 10$, $\Delta = 15$, $p = 0.9$.

3 General Model

In this section we show the general model for the execution of N tasks on M slaves. We also show a specific instance for 2 slaves and we show its solution.

3.1 Preliminary Results

The Markov chain-based model for the general case is shown in Fig. 11. At the beginning of the computation, n task are scheduled to the m resources. The next state depends on the number of failures/successes:

- If m failures happen, we stay in the initial state. The probability that this happens is equal to $q^m$. The time spent in this operation is equal to $\Delta$, and the time needed to terminate the computation is equal to the execution of n tasks on m resources, as at the beginning.

- The cases of 1 to m-1 failures make the state transit in states from $n - m + 1$ to 1, respectively. The probability of each transition is computed exploiting a binomial expression, properly multiplied for the probabilities. The time needed to perform the operation is the maximum between $\delta$ and $\Delta$ (i.e. $\mu$). The time needed to complete the computation depends on the amount of remaining tasks.

- If no failures happen, we transit in the state labeled with $n - m$. The probability of such a transition is $p^m$, with time equal to the average task execution time. The time needed to complete the computation, in this case, is equal to the computation of $n - m$ tasks.
Table 5: Results of 100 tests of execution of 2 tasks on 2 interpreters, with $\delta = 10$, $\Delta = 15$, $p = 0.2$.

The general model can be represented as a single recurrence over the completion times:

$$
\tau_{n,m} = q^m(\Delta + \tau_{n,m}) + \sum_{k=1}^{m-1} \binom{m}{k} p^k q^{m-k}(\mu + \tau_{n-k}) + p^m(\delta + \tau_{n-m,m})
$$

The first part of the expression is related to $m$ failures. This takes $\Delta$ time, i.e. the time of failure detection plus the resource restart, and the additional time needed to perform the whole computation, as we have still to perform all tasks. The second part of the expression is related to 1 to $m-1$ failures. It takes $\mu$ time, depending on the actual $\delta$ and $\Delta$ values, plus the time needed to perform the rest of the computation. The last part is related to $M$ successes, that takes $\delta$ time, plus the time needed to perform the rest of the computation.

It is interesting to show how the general model can be instantiated (Fig. 12). For the case of $N$ tasks performed on 2 slaves, we show the linear recurrence expression, and its solution. The mathematical steps we show are only the main ones, according to a standard methodology to solve linear recurrences. The recurrence expression for $N$ tasks performed by 2 slaves is:

$$
\tau_n = q^2 \cdot (\Delta + \tau_n) + 2pq \cdot (\mu + \tau_{n-1}) + p^2 \cdot (\delta + \tau_{n-2})
$$

This formula can be expressed in the classical form of linear recurrences as:

$$
\tau_n = \frac{q^2}{1-q^2} \tau_{n-1} \quad \frac{2pq}{1-q^2} \tau_{n-2} + \frac{q^2}{1-q^2} \Delta \quad \frac{2pq}{1-q^2} \mu + \frac{p^2}{1-q^2} \delta
$$

To solve the recurrence, three main steps can be followed:
Table 6: Results of 100 tests of execution of 2 tasks on 2 interpreters, with $\delta = 10$, $\Delta = 15$, $p = 0.5$.

1. Solve the homogeneous part of the recurrence.
2. Find particular solutions.
3. Substitute boundary conditions in the general solution to develop it.

Below, we shortly show each step.

**Solve Homogeneous**  The homogeneous part of the linear recurrence is:

$$(1 - q^2)\tau_n - 2pq\tau_{n-1} - p^2\tau_{n-2}$$

and we impose it to be equal to zero:

$$(1 - q^2)\alpha^2 - 2pq\alpha - p^2 = 0$$

We solve it, and we obtain two solutions:

$$\alpha = \begin{cases} 1 \\ \frac{q - 1}{q + 1} \end{cases}$$

Thus, the homogeneous solution has the following form:

$$\gamma_n = A \left( \frac{q - 1}{q + 1} \right)^n + B(1)^n$$
Table 7: Discrepancies between monitored and theoretical completion times of 2 Tasks executed on 2 Interpreters. Values are characterized w.r.t. the $\delta < \Delta$ condition.

<table>
<thead>
<tr>
<th>Monitored</th>
<th>Theoretically Computed</th>
<th>Discrepancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta &gt; \Delta$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.37679</td>
<td>12</td>
<td>0.62321</td>
</tr>
<tr>
<td>12.83438</td>
<td>13.95833</td>
<td>1.12395</td>
</tr>
<tr>
<td>20.14205</td>
<td>21.66667</td>
<td>1.52465</td>
</tr>
<tr>
<td><strong>Average Discrepancy</strong></td>
<td><strong>1.091</strong></td>
<td></td>
</tr>
<tr>
<td>$\delta &lt; \Delta$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.07644</td>
<td>13.18182</td>
<td>0.10538</td>
</tr>
<tr>
<td>15.21683</td>
<td>16.875</td>
<td>1.65817</td>
</tr>
<tr>
<td>26.57598</td>
<td>35</td>
<td>9.42402</td>
</tr>
<tr>
<td><strong>Average Discrepancy</strong></td>
<td><strong>3.729</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Total Average Discrepancy</strong></td>
<td><strong>2.429</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Discrepancies between monitored and theoretical completion times of 2 Tasks executed on 2 Interpreters. Values are characterized w.r.t. the failure probability.

<table>
<thead>
<tr>
<th>Monitored</th>
<th>Theoretically Computed</th>
<th>Discrepancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p = 0.1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11.37679</td>
<td>12</td>
<td>0.62321</td>
</tr>
<tr>
<td>13.07644</td>
<td>13.18182</td>
<td>0.10538</td>
</tr>
<tr>
<td><strong>Average Discrepancy</strong></td>
<td><strong>1.091</strong></td>
<td></td>
</tr>
<tr>
<td>$p = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12.83438</td>
<td>13.95833</td>
<td>1.12395</td>
</tr>
<tr>
<td>15.21683</td>
<td>16.875</td>
<td>1.65817</td>
</tr>
<tr>
<td><strong>Average Discrepancy</strong></td>
<td><strong>1.091</strong></td>
<td></td>
</tr>
<tr>
<td>$p = 0.5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20.14205</td>
<td>21.66667</td>
<td>1.52465</td>
</tr>
<tr>
<td>26.57598</td>
<td>35</td>
<td>9.42402</td>
</tr>
<tr>
<td><strong>Average Discrepancy</strong></td>
<td><strong>1.091</strong></td>
<td></td>
</tr>
</tbody>
</table>

Find Particulars  Typically, to obtain a particular solution, we guess one of the same or higher degree of the inhomogeneous, and we verify it is correct. We have found that

$$\tau_n = cn$$

is a particular solution, where:

$$c = \frac{q^2\Delta + 2pq\mu + q^2\delta}{2p}$$

Obtain General Solution  The general solution has the form:

$$\tau_n = \text{homogeneous solution} + \text{particular solution}$$

That is:

$$\tau_n = A \left( \frac{q - 1}{q + 1} \right)^n + B + \frac{q^2\Delta + 2pq\mu + q^2\delta}{2p}$$

We can substitute the following boundary conditions:

$$t_1 = p\delta + q\Delta$$
Figure 11: Markov chain modeling the execution of \( n \) task on \( m \) interpreter, in the case of failure/restart of the computing resources.

and

\[
t_2 = p^2 \delta + 2pq\mu + q^2 \Delta
\]

We can rework the solution in a way that it depends only on \( A \):

\[
\tau_n = A \left( \frac{q - 1}{q + 1} \right)^n + \frac{t_2}{2p} n - A \left( \frac{q - 1}{q + 1} \right)^2 - \frac{q}{1 - q^2} (t_2 - 2t_1)
\]

By substituting the boundary conditions \( t_1 \) and \( t_2 \) we obtain:

\[
A = \frac{1}{1 - q^2} \cdot \left( \frac{3 + q}{2} \cdot t_2 - 2qt_1 \right) - \left( 1 + \frac{2pq}{1 - q^2} \right) \cdot \frac{t_2}{1 - q^2} - 1 \cdot \frac{1}{1 - q^2} \left( \frac{2pq}{1 - q^2} \cdot 2pq + p^2 \right) \cdot \frac{t_1}{p}
\]

3.2 Future Work

The evolution of the study in the context of the collaboration between UPC and UNIPI institutes will be based on the following guidelines:

- Studying further instances of the general case, like \( N \) tasks performed on 3, 4, etc ... interpreters. This study should help in acquiring knowledge on the form of the general solution.

- By exploiting the results of the previous point, studying directly the solutions for the general case for \( N \) tasks on \( M \) resources.

- Explore other techniques to solve the general recurrence model. For instance, in [10] it is presented a methodological approach to study the expected running time of generic computations that can be described as absorbing Markov chains. In this case, the problem resides in the fact that it is not possible to write a complete and closed matrix (describing the Markov chain) of the general case.

- Analyze other fault tolerance techniques. For example, hot redundancy for task execution. For this last case, the model seems much more simple (than the one we presented in this document) locally on each set of interpreters performing the same tasks. Anyway, the general model presents the same problems of the one described in this document.

The work will include both theoretical analysis and experiments, in muskel, and in other programming models.
4 Conclusions

In this technical document we have shown the results of the study of quantitative aspects in fault-tolerant task parallel computations. In particular, we addressed task re-scheduling as fault tolerance technique, and we exploited Markov models to study the faulty performance, i.e. the performance of parallel computations in the case faults happen by following an exponential distribution. The results include quantitative general models for task parallel computations, and solutions for base cases and an instance of the general model.

These results and their continuations will be included as part of a PhD thesis of the University of Pisa, by Carlo Bertolli. The results, along with their conclusive parts, will be also object of publications in the near future. Some results included in this document have been proposed for the CoreGrid Integration Workshop in 2008.

The work has been done in the context of the CoreGrid project, as a collaboration between the UPC and UNIPI institutes. The collaboration between the two institutes matured thanks to their long histories in the context of parallel computing. More specifically the two groups studied in depth the research fields related to parallel programming languages, their supports, and the quantitative analysis of parallel algorithms.

The collaboration started from the visit of Prof. Peter Kilpatrick at the University of Pisa (REP 4), and that of Prof. Joaquim Gabarró (REP 12). During this visit the experience on theoretical computer science of Prof. Kilpatrick and Prof. Joaquim Gabarró motivated our groups to start a new joint work addressing theoretical, analytical and implementation-related studies, specifically related to the field of adaptive and autonomic high-performance computing.

References


