European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies

A Network of Excellence funded by the European Commission

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

CoreGRID TR-TR-0114
September 30st, 2007

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 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. $out = F_1(F_2(\ldots F_n(in)\ldots))$), and the results are delivered to an output stream. Each function 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

```
1 Compute worker1 = new Inc();
2 Compute stage1 = new Farm(worker1);
3 Compute worker2 = new Inc();
4 Compute stage2 = new Farm(worker2);
5 Compute main = new Pipeline(stage1,stage2);
```

Figure 1: Example of the definition of the macro data-flow graph in a `muskel` program.

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 δ. 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 Δ_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.

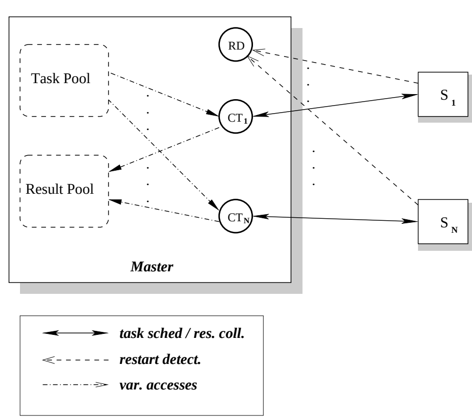

Figure 2: Graphical representation of the implementation of the FT strategy of `muskel` .

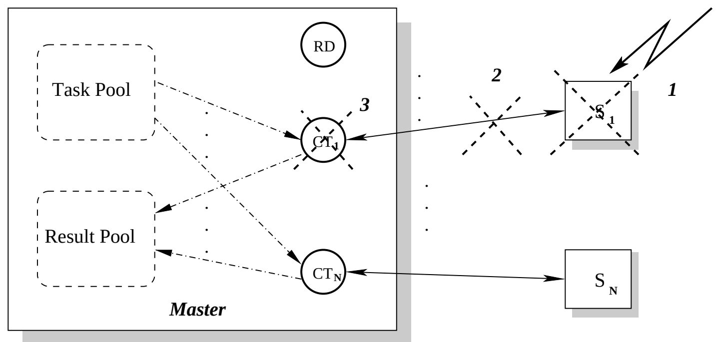

Figure 3: Graphical representation of the master behavior in the case of slave failure. When the slave fails (1), the connection with the corresponding control thread is closed (2). The control thread detects the closing of the connection and it terminates itself (3).

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. Δ_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

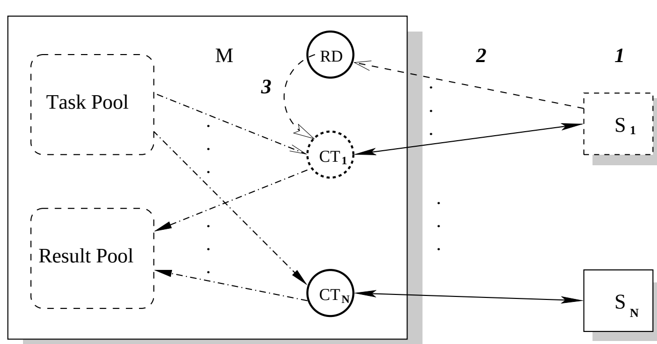

Figure 4: Graphical representation of the master behavior in the case of slave restart. When slave S_1 is restarted by some external mechanism (1), and it communicates the RD thread on the master of its availability (2). RD spawns a new control thread for the restarted slaves (3).

```
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.

responsible of scheduling tasks to interpreters and, in the case of failure, it re-contacts the new restarted interpreter (bold arrow). 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 Δ_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: $t = \lceil \frac{T_C}{T_T} \rceil$. 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 Δ_F and Δ_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).

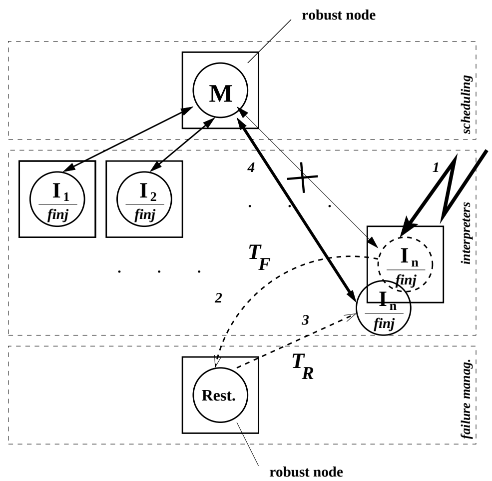

Figure 6: Representation of the first instances of the fault injection technique. Whenever a slave fails (1) it detects the failure in T_F time (2), and it re-spawns it (3) in T_R time (dotted arrows).

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 δ the average time needed to perform a task. We denote with Δ 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 Δ_F); (c) restart the slave (denoted with Δ_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 μ the maximum between δ and Δ, 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

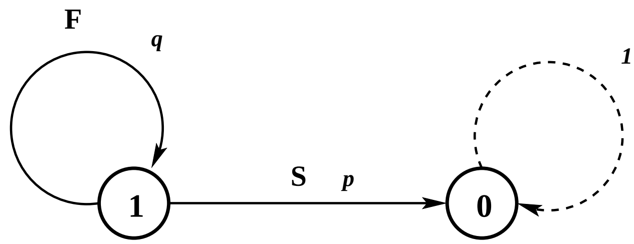

Figure 7: Markov chain modeling the execution of 1 task on 1 interpreter, in the case of failure/restart of the computing resources.

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 δ 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 Δ 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 $\frac{1}{p}$. We can compute the average time needed to execute the whole computation as:

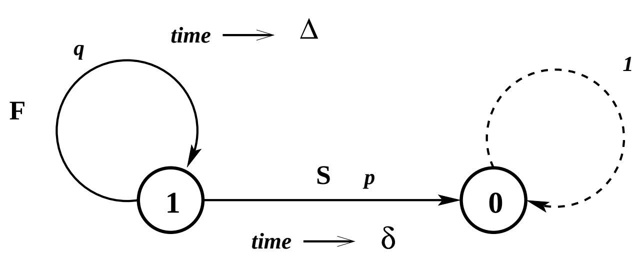

Figure 8: Markov chain modeling the execution of 1 task on 1 interpreter augmented with timing information. Δ represents the average time needed to: (a) perform part of the task, (b) detect a failure, and (c) restart the corresponding interpreter. δ represents the average time needed to perform a task.

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

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. Δ time for each failure, and δ time for the last successful execution). We can divide the expression in two terms by multiplying the sums:

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

and we can take a λp term and a Δp 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:

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

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:

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

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 a 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 Δ 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 δ 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 Δ 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 δ time.

Both states labeled with 0 are absorbing.

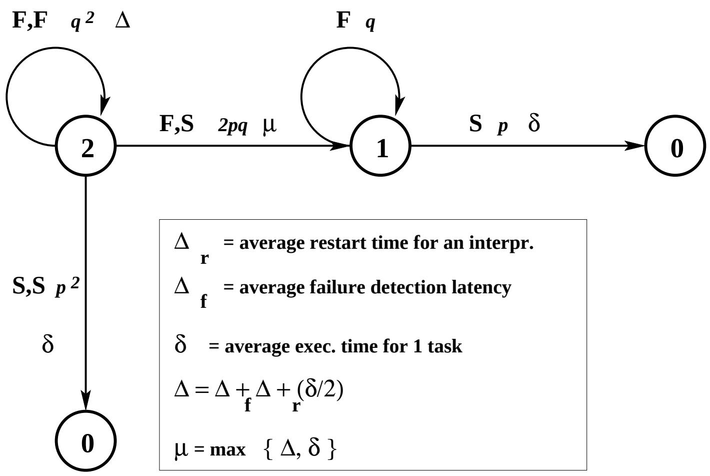

Figure 9: Markov chain modeling the execution of 2 tasks on 2 interpreters.

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 $q^{2 \cdot k} \cdot 2pq$. 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:

$$\sum_{(w_1, w_2)} t(w_1) \cdot p(w_1) \cdot p(w_2) + \sum_{(w_1, w_2)} t(w_2) \cdot p(w_1) \cdot p(w_2)$$

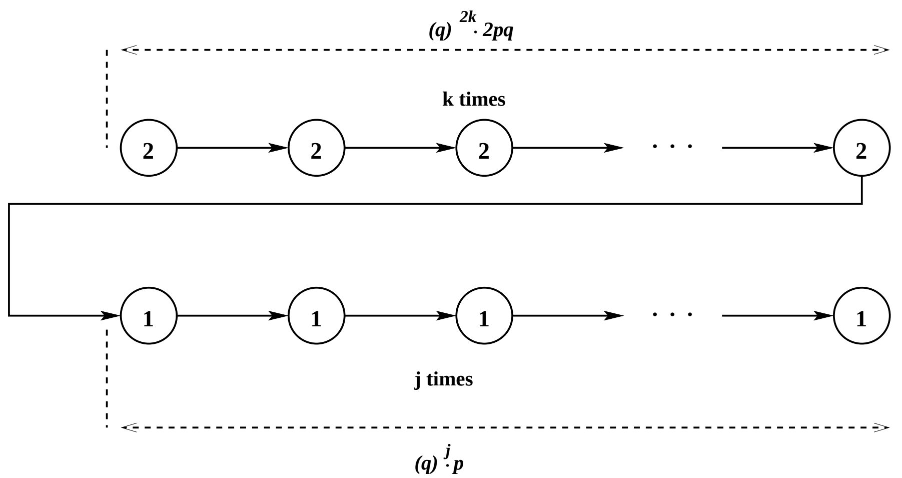

Figure 10: Scheme of the behavior of retries for the longest path, in the case of one failure and one success. In this behavior we first iterate on the state 2 (with 2 failures), next we obtain a success and a failure and we transit in the state 1. The state 1 is iterated until a success is obtained.

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

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

The two sums for single probabilities can be solved:

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

and

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

The whole upper bound on the expected time is:

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

In the next subsections we consider the three quantities independently.

First Sub-Expression We resolve the expression:

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

We perform the multiplication of Δ and μ 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 $\frac{pq}{1-q^2}$ 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:

$$\sum_{w_2=1\rightsquigarrow 0} t(w_2) \cdot p(w_2) = 2\mu \frac{pq}{1-q^2} + 2pq\Delta \frac{q^2}{(1-q^2)^2}$$

Second Sub-Expression We resolve the expression:

$$\sum_{w_2=1\rightsquigarrow 0} t(w_2) \cdot p(w_2)$$

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:

$$\sum_{w_2=1\rightsquigarrow 0} t(w_2) \cdot p(w_2) = \frac{2pq}{1-q^2}[\delta + \Delta \frac{q}{p}]$$

Third Sub-Expression We resolve the expression:

$$\sum_{w_3=2\rightsquigarrow 0} t(w_3) \cdot p(w_3)$$

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:

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

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:

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

Finally:

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

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 ∞. We consider the time and the probability of each path, and we compute the variance as:

$$Var[X] = E[X^2] - E^2[X]$$

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

$$\sum_w \tau^2(w)p(w) = \sum_{2 \rightsquigarrow 1 \rightsquigarrow 0} \tau^2(w)p(w) + \sum_{2 \rightsquigarrow 0} \tau^2(w)p(w)$$

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:

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

We develop as follows:

$$\begin{aligned}
&\left[(\Delta k_1 + \mu + \Delta k_2 + \delta)^2\right] \cdot p(w) \\
&\quad = \left[((\Delta k_1 + \Delta k_2) + (\mu+\delta))^2\right] \cdot p(w) \\
&\quad = \left[(\Delta k_1 + \Delta k_2)^2 + 2(\Delta k_1 + \Delta k_2)(\mu+\delta) + (\mu+\delta)^2)\right] \cdot p(w) \\
&\quad = \left[\Delta^2 k_1^2 + 2\Delta^2 k_1 k_2 + \Delta^2 k_2^2 + 2\Delta k_1(\mu+\delta) + 2\Delta k_2(\mu+\delta) + (\mu+\delta)^2\right] \cdot p(w)
\end{aligned}$$

We consider each term independently and we obtain:

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

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

$$C = \sum_{k_1,k_2} \Delta^2 k_2^2 (q^2)^{k_1} 2pqq^{k_2} p = 2\Delta^2 \cdot \frac{q^2}{p^2}$$

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

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

$$F = \sum_{k_1,k_2} (\mu+\delta)^2 (q^2)^{k_1} 2pqq^{k_2} p = (\mu+\delta)^2 \frac{q}{1+q}$$

Clearly, the first path can be computed as:

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

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:

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

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

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

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

$$\sum_{2\rightsquigarrow 0} \tau^2(w)p(w) = G+H+I$$

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:

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

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 μ, δ or Δ.

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:

$$E = \frac{1}{n} \sum_{x=1}^{n} T^x_{comp}$$

$$Var = \frac{1}{n} \sum_{x=1}^{n} (T^x_{comp} - E)^2$$

Where $n = 100$, T_{comp} is the set of the obtained completion times, and T^i_{comp} 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

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

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

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

Resolving the equation:

$$T_{comp} \leq 12$$

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).

Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}
1	10.464	2	10.372	3	17.824	4	10.368
5	10.36	6	10.39	7	10.364	8	10.344
9	10.357	10	10.357	11	10.371	12	10.377
13	10.357	14	13.251	15	10.402	16	20.384
17	10.348	18	12.229	19	10.389	20	10.372
21	10.372	22	10.363	23	20.399	24	10.379
25	10.368	26	10.369	27	20.382	28	10.384
29	10.358	30	10.361	31	10.361	32	10.375
33	10.357	34	10.355	35	10.355	36	10.368
37	10.367	38	18.86	39	10.352	40	10.364
41	10.357	42	20.433	43	10.385	44	10.337
45	10.358	46	10.369	47	14.274	48	10.409
49	17.817	50	10.36	51	10.363	52	10.361
53	17.318	54	10.385	55	10.355	56	10.363
57	10.369	58	10.362	59	10.367	60	10.375
61	10.361	62	16.812	63	10.362	64	10.367
65	10.382	66	10.371	67	10.363	68	10.368
69	16.819	70	10.379	71	10.362	72	10.363
73	10.348	74	10.352	75	10.374	76	10.366
77	10.361	78	10.357	79	10.362	80	10.365
81	10.36	82	10.381	83	10.361	84	10.369
85	10.358	86	10.375	87	10.353	88	19.337
89	10.352	90	10.36	91	10.367	92	10.362
93	10.372	94	10.357	95	10.372	96	10.36
97	10.354	98	10.35	99	10.373	100	10.366
average	**11.37679**	**variance**	**7.229**				

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:

$$T_{comp} \leq 13.95833$$

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:

$$T_{comp} \leq 21.66667$$

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).

Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}
1	19.844	2	10.378	3	10.356	4	10.367
5	25.502	6	10.69	7	23.8	8	10.381
9	16.81	10	10.385	11	20.403	12	10.331
13	20.403	14	10.399	15	10.287	16	10.355
17	10.357	18	10.356	19	13.174	20	10.416
21	10.392	22	10.369	23	10.363	24	20.392
25	10.389	26	18.349	27	20.373	28	10.37
29	10.379	30	10.289	31	10.358	32	18.862
33	10.367	34	23.931	35	10.695	36	10.405
37	10.367	38	30.205	39	10.39	40	17.306
41	10.357	42	10.364	43	18.332	44	10.416
45	10.342	46	18.838	47	10.287	48	14.27
49	10.393	50	10.303	51	10.369	52	10.363
53	10.376	54	10.366	55	10.379	56	10.366
57	10.356	58	10.359	59	10.361	60	16.818
61	10.373	62	10.359	63	10.315	64	10.35
65	10.36	66	19.869	67	10.385	68	10.365
69	10.332	70	10.363	71	10.362	72	10.382
73	11.135	74	10.405	75	10.362	76	10.372
77	20.786	78	10.371	79	24.517	80	10.372
81	10.362	82	10.367	83	10.371	84	10.353
85	20.387	86	10.376	87	10.36	88	10.374
89	25.994	90	10.372	91	10.349	92	10.373
93	10.373	94	10.344	95	20.407	96	10.374
97	10.362	98	10.365	99	13.76	100	11.72
average	**12.83438**	**variance**	**21.542**				

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 $\delta < Delta$, 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:

$$T_{comp} \leq 16.875$$

Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}
1	34.142	2	41.218	3	10.399	4	36.212
5	10.369	6	10.363	7	17.84	8	10.362
9	10.353	10	10.364	11	17.821	12	10.383
13	10.361	14	12.739	15	10.398	16	34.735
17	35.237	18	10.373	19	29.362	20	33.124
21	39.355	22	36.768	23	19.854	24	10.365
25	10.373	26	10.354	27	13.76	28	19.863
29	15.785	30	14.208	31	10.376	32	10.349
33	10.369	34	10.388	35	27.216	36	11.722
37	25.482	38	10.386	39	20.384	40	15.286
41	13.25	42	10.405	43	25.985	44	42.394
45	12.733	46	17.308	47	13.757	48	12.228
49	24.981	50	30.057	51	10.366	52	20.396
53	10.397	54	12.23	55	14.767	56	37.681
57	32.961	58	10.319	59	24.905	60	10.386
61	20.394	62	33.633	63	30.582	64	11.202
65	28.629	66	11.192	67	37.136	68	15.791
69	51.818	70	25.423	71	10.387	72	21.829
73	27.527	74	18.329	75	31.09	76	21.92
77	11.718	78	38.154	79	11.718	80	10.483
81	10.351	82	25.06	83	10.412	84	28.88
85	10.374	86	24.974	87	27.063	88	32.05
89	22.436	90	10.361	91	10.374	92	53.783
93	10.363	94	13.762	95	11.211	96	32.113
97	34.041	98	10.36	99	30.564	100	10.414
average	**20.14205**	**variance**	**116.982**				

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: $\delta = 10$, $\Delta = \Delta_F + \Delta_R = 1 + 14 = 15$, $\mu = max\delta, \Delta = \Delta = 15$, $p = 0.5$, $q = 0.5$.

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

$$T_{comp} \leq 35$$

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.

Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}
1	10.6	2	10.342	3	10.376	4	21.917
5	10.409	6	10.366	7	10.355	8	23.448
9	10.427	10	10.365	11	10.378	12	10.337
13	23.448	14	10.4	15	10.365	16	10.366
17	10.365	18	10.367	19	20.392	20	20.908
21	10.431	22	10.282	23	19.879	24	10.505
25	10.291	26	18.859	27	17.836	28	10.424
29	10.346	30	10.381	31	10.348	32	10.372
33	10.366	34	10.29	35	10.359	36	10.414
37	21.399	38	10.375	39	10.365	40	22.944
41	10.334	42	10.349	43	10.372	44	20.9
45	20.395	46	10.371	47	10.35	48	19.862
49	10.375	50	22.431	51	10.468	52	10.368
53	10.366	54	10.362	55	10.356	56	10.356
57	10.386	58	15.799	59	22.433	60	10.384
61	10.382	62	10.364	63	10.367	64	10.354
65	10.368	66	10.406	67	10.344	68	10.339
69	10.357	70	24.462	71	23.96	72	25.055
73	17.841	74	10.402	75	10.359	76	10.385
77	10.362	78	20.408	79	10.362	80	10.358
81	10.376	82	10.353	83	10.369	84	10.376
85	10.355	86	10.357	87	10.341	88	10.367
89	24.473	90	10.4	91	10.358	92	10.383
93	10.365	94	10.366	95	10.367	96	20.386
97	10.384	98	10.358	99	18.342	100	21.919
average	**13.07644**	**variance**	**23.261**				

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 Δ, 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 δ and Δ (i.e. μ). 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.

Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}
1	10.528	2	10.373	3	10.363	4	10.402
5	24.474	6	10.406	7	24.981	8	23.449
9	10.412	10	19.369	11	10.383	12	10.354
13	10.367	14	10.387	15	10.353	16	10.369
17	10.43	18	35.68	19	10.401	20	10.365
21	10.298	22	20.383	23	32.642	24	10.392
25	10.368	26	10.351	27	25.06	28	15.291
29	16.818	30	15.29	31	10.385	32	28.054
33	10.39	34	31.609	35	10.39	36	16.311
37	17.843	38	10.391	39	10.363	40	23.45
41	22.426	42	10.466	43	10.366	44	18.862
45	23.958	46	10.483	47	10.349	48	10.361
49	20.393	50	10.372	51	10.353	52	10.358
53	20.898	54	10.391	55	10.372	56	10.359
57	20.393	58	21.918	59	23.444	60	10.411
61	20.899	62	10.402	63	10.373	64	24.471
65	36.269	66	23.424	67	16.293	68	10.376
69	10.365	70	10.382	71	21.406	72	10.382
73	10.364	74	10.368	75	15.29	76	20.389
77	10.396	78	10.36	79	10.358	80	10.355
81	10.361	82	10.359	83	10.369	84	22.438
85	17.832	86	10.386	87	16.82	88	22.939
89	10.407	90	10.361	91	10.361	92	10.283
93	23.451	94	18.341	95	21.918	96	16.82
97	16.802	98	20.897	99	10.405	100	10.353
average	**15.21683**	**variance**	**42.973**				

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:

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

The first part of the expression is related to m failures. This takes Δ 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 μ time, depending on the actual δ and Δ values, plus the time needed to perform the rest of the computation. The last part is related to M successes, that takes δ 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:

$$\tau_n = \underbrace{\frac{q^2}{1-q^2}\tau_{n-1} + \frac{2pq}{1-q^2}\tau_{n-2}}_{homogeneous\ part} + \underbrace{\frac{q^2}{1-q^2}\Delta + \frac{2pq}{1-q^2}\mu + \frac{p^2}{1-q^2}\delta}_{inhomogeneous\ part}$$

To solve the recurrence, three main steps can be followed:

Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}	Num.	T_{comp}
1	49.022	2	10.389	3	22.425	4	10.456
5	27.613	6	17.822	7	10.388	8	10.363
9	10.364	10	23.961	11	24.468	12	10.418
13	36.211	14	27.023	15	101.549	16	19.879
17	10.378	18	23.452	19	10.384	20	38.237
21	20.899	22	16.744	23	15.799	24	16.818
25	20.899	26	10.414	27	34.75	28	21.915
29	19.368	30	54.127	31	20.388	32	22.937
33	23.959	34	34.732	35	19.369	36	15.29
37	19.872	38	23.452	39	60.757	40	44.436
41	43.931	42	10.399	43	22.935	44	10.385
45	42.82	46	10.379	47	54.648	48	19.352
49	23.96	50	10.381	51	23.452	52	10.404
53	34.16	54	20.404	55	23.448	56	53.109
57	21.418	58	33.728	59	10.38	60	20.419
61	34.742	62	10.388	63	10.402	64	10.356
65	34.241	66	16.238	67	25.561	68	10.392
69	10.358	70	15.291	71	23.955	72	18.842
73	51.585	74	34.148	75	34.147	76	10.373
77	60.243	78	10.392	79	58.202	80	10.419
81	30.073	82	25.498	83	61.694	84	20.408
85	23.451	86	81.16	87	24.469	88	18.842
89	21.925	90	22.937	91	20.395	92	48.523
93	22.866	94	10.399	95	21.408	96	17.33
97	79.097	98	37.213	99	32.539	100	35.257
average	**26.57598**	**variance**	**295.833**				

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:

$$\gamma_n = A\left(\frac{q-1}{q+1}\right)^n + B(1)^n$$

Monitored	Theoretically Computed	Discrepancy
$\delta > \Delta$		
11.37679 12.83438 20.14205	12 13.95833 21.66667	0.62321 1.12395 1.52465
Average Discrepancy	**1.091**	
$\delta < \Delta$		
13.07644 15.21683 26.57598	13.18182 16.875 35	0.10538 1.65817 9.42402
Average Discrepancy	**3.729**	
Total Average Discrepancy	**2.429**	

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.

Monitored	Theoretically Computed	Discrepancy
$p = 0.1$		
11.37679 13.07644	12 13.18182	0.62321 0.10538
Average Discrepancy	**1.091**	
$p = 0.2$		
12.83438 15.21683	13.95833 16.875	1.12395 1.65817
Average Discrepancy	**1.091**	
$p = 0.5$		
20.14205 26.57598	21.66667 35	1.52465 9.42402
Average Discrepancy	**1.091**	

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.

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:

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

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:

$$t_1 = p\delta + q\Delta$$

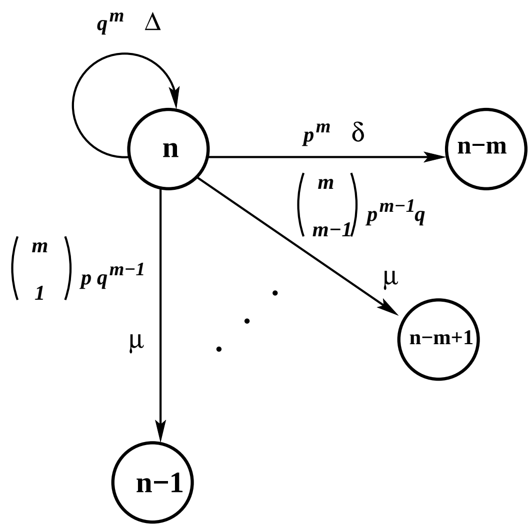

Figure 11: Markov chain modeling the execution of n task on m interpreter, in the case of failure/restart of the computing resources.

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:

$$A = \frac{\frac{1}{1-q^2}\cdot\left(\frac{3+q}{2}\cdot t_2 - 2qt_1\right) - \left(1+\frac{2pq}{1-q^2}\right)\cdot\frac{t_2}{1-q^2} - \frac{1}{1-q^2}\left(\frac{2pq}{1-q^2}\cdot 2pq + p^2\right)\cdot\frac{t1}{p}}{\left(\frac{q-1}{q+1}\right)^2\cdot\frac{2}{(q+1)}}$$

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 aquiring 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.

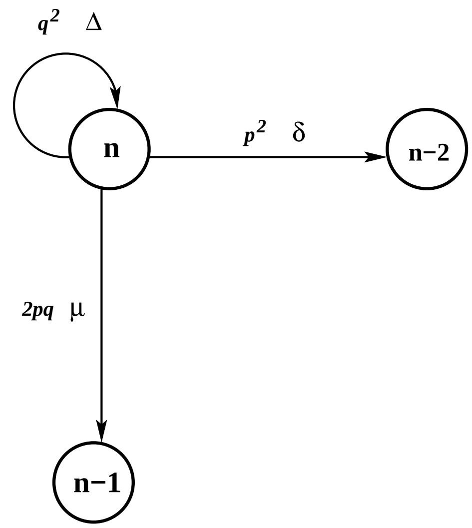

Figure 12: Markov chain of the model for N tasks performed on 2 Interpreters.

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 maturated 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

[1] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, J. González, C. León, L. Moreno, J. Petit, J. Roda, A. Rojas, and F. Xhafa. Mallba: A library of skeletons for combinatorial optimisation. In B. Monien and R. Feldman, editors, *Euro-Par 2002 Parallel Processing*, volume 2400 of *Lecture Notes in Computer Science*, pages 927–932. Springer-Verlag, Berlin Heidelberg, 2002.

[2] E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, and F. Xhafa. Efficient parallel lan/wan algorithms for optimization: the mallba project. *Parallel Comput.*, 32(5):415–440, 2006.

[3] Marco Aldinucci and Marco Danelutto. Algorithmic skeletons meeting grids. *Parallel Computing*, 32(7-8):449–462, 2006. DOI:10.1016/j.parco.2006.04.001.

[4] Carlo Bertolli, Massimo Coppola, and Corrado Zoccolo. The co-replication methodology and its application to structured parallel programming. In *Proc of. HPC-GECO/Compframe (to appear)*, October 2007.

[5] Robert D. Blumofe and Philip A. Lisiecki. Adaptive and reliable parallel computing on networks of workstations. In *ATEC'97: Proceedings of the Annual Technical Conference on Proceedings of the USENIX 1997 Annual Technical Conference*, pages 10–10, Berkeley, CA, USA, 1997. USENIX Association.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[7] Murray Cole. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Comput.*, 30(3):389–406, 2004.

[8] Ian Foster and Carl Kesselman, editors. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[9] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Syst.*, 1(3):222–238, 1983.

[10] Kishar Shridharbhai Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1982.

[11] Marco Vanneschi. The programming model of assist, an environment for parallel and distributed portable applications. *Parallel Comput.*, 28(12):1709–1732, 2002.