Open Journal of Optimization, 2021, 10, 71-87
https://www.scirp.org/journal/ojop
ISSN Online: 2325-7091
ISSN Print: 2325-7105

# Model for Anticipating Failures by Omission in Calculation Grids

**Ramadane Adamou Yougouda**[1], **Marcellin Nkenlifack**[2], **Vivient Corneille Kamla**[1], **Laurent Bitjoka**[1]

[1]ENSAI, University of Ngaoundere, Ngaoundere, Cameroon
[2]URIFIA, Department of Mathematics and Computer Science, Faculty of Science, University of Dschang, Dschang, Cameroon
Email: ramadane.adamou@univ-ndere.cm

**How to cite this paper:** Yougouda, R.A., Nkenlifack, M., Kamla, V.C. and Bitjoka, L. (2021) Model for Anticipating Failures by Omission in Calculation Grids. *Open Journal of Optimization*, **10**, 71-87.
https://doi.org/10.4236/ojop.2021.103006

**Received:** June 15, 2021
**Accepted:** August 29, 2021
**Published:** September 1, 2021

## Abstract

Computer grids are infrastructures in which heterogeneous and distributed resources offer very high computing or storage performance. If they offer extreme computing performance, they are also subject to the appearance of many failures related to this type of architecture. While performing tasks, if the response time of a node in the system incomprehensibly exceeds the requirements of the specifications, the node experiences an omission failure. The task running in the failed node will be unavailable until the node resumes normal activity. Waiting not being a possible solution, many fault tolerance methods have been proposed. Despite this large number of fault tolerance methods on offer, computer grids are still prone to many failures by omission. In this work, a numerical study of the failures by omission which occur in the calculation grids during the execution of the tasks was carried out and a model allowing anticipating its failures was proposed with the formalism PDEVS (Parallel Discret EVent system Specification).

## Keywords

Calculation Grids, Fault Tolerance, Failures by Omissions, PDEVS, DEVS, Modeling, Simulation, DEVSJAVA

## 1. Introduction

The need to have more and more computing or storage power for projects has pushed humans to create increasingly efficient infrastructures [1]. Since the appearance of the first computers, passing by supercomputers, then clusters to computer grids, the objective has always been the search for ever higher performance. Computer networks are today the most efficient infrastructures in terms of power [2]. IT grid is a virtual infrastructure made up of a set of potentially shared, distributed, heterogeneous, delocalized and autonomous IT resources. This distributed, heterogeneous and delocalized architecture that computer grids possess is at the origin of many failures, including failures by omission [3]. A failure known as "by omission" is a failure in which the component temporarily ceases its activity and then resumes its normal activity [4]. The origins of failures by omission are numerous, we can cite: The increase in processor temperature because when the processor temperature approaches the maximum temperature allowed on the processor's Integrated Heat Distributor (IHS) is also referred to as the chassis temperature. Congestion when the RAM occupancy rate approaches 100%. Many fault tolerance methods have been implemented so that the computer network can function despite the presence of faults [5] [6] [7]. There are several fault tolerance methods that are proposed in the literature [6]:

1) Fault tolerance based on the replication method. Different works have presented several replication methods: active, passive, semi-active, coordinator/cohort and adaptive. All of these methods consist of creating a set of replicas in different nodes. These replicas communicate with each other and with the system. When a failure occurs, one of the different replicas is chosen after consensus to return the result of the final processing [8]. But the fact that each of the replicas uses the resources and decreases the overall power of the grid, is not a suitable method for computing grids which require the maximum of resources.

2) Recovery-based fault tolerance. Two categories of restoration-based fault tolerance methods are proposed in the literature: backward recovery which consists of making system backups and, in the event of node failure, restoring from the last backup. And recovery by pursuit, which consists in the event of a failure of a node, reconstructs the failed job from a consensus between the other jobs [4].

Despite the evolution of fault tolerance methods, there are still many failures that occur in the performance of tasks in computer grids [9] [10]. For us, therefore, it is a question of answering the following question "Can we not propose a model that can anticipate failures by omission?". The goal of this article is to propose a model which makes it possible to anticipate failures by omission in the calculation grids during the execution of the tasks. The plan for this article is as follows. In Section 2, we will present the corrected and failing models of a computing grid and in Section 3, we will present the models that allow anticipating failures by omission in computing grids.

## 2. Modeling and Simulation of a Computer Grid

To model and simulate omission failures in computational grids, we use the PDEVS formalism to represent 100 nodes. In the following we present a calculation grid and the formalism used

### 2.1. Description of a Computer Grid

Ian Foster and Carl Kesselman were the first to use the word "grid" in their book "The Grid: Blueprint for a New Computing infrastructure" published in 2003 [11]. They compare a computer network to an electrical network in terms of the availability of resources. In an electrical network, all you have to do is plug an outlet into an outlet and electricity is supposed to flow. In a computer grid, it is enough to connect to the network to have the resources of the grid. a computer grid is defined as a set of shared, distributed, heterogeneous, delocalized and autonomous hardware and software resources in which large-scale scientific problems can be solved [6] [12]. The life cycle of a job in a grid IT begins when the user obtains a certificate from a certificate authority trusted by the grid. The user then registers in a virtual grid organization and obtains their user certificate [13]. The user can then connect to the platform from a particular machine acting as a user interface from which he can launch a job. This job is submitted from the user interface to the job management system (WMS) through its "tache" input port. The WMS searches for the calculation elements (CE) or nodes available to process the job. The system if job finds the grid at rest, ie in a "attente" state, it goes into an "active" state during which it processes the job. If the job finds the system in an "active" state, the job is just waiting for its turn to be processed. If a fault by omission is committed during the execution of the job. For example, jobs are sent to a node faster than it can process them, the node gets congested and an omission failure occurs on that node and its state goes to omission until the node decongests itself [14]. Once the processing of the parts of the job is finished, the WMS reconstructs the processed task and the processed job is returned by the "error" port [12].

## 2.2. The PDEVS Formalism

To model and simulate failures by omission in computing grids, we used the PDEVS formalism. A computing grid being a very complex autonomous system seen as a single computer but which is a connection of several computers which share their resources in order to provide a very high power [15]. This modeling cannot be done with standard modeling formalisms because the latter are less suited to the modeling of complex and autonomous systems [9]. The modeling was done with the PDEVS modeling formalism. The PDEVS formalism allows modeling of causal and deterministic systems [10]. Its role is to provide a simple technique of parallelization of calculations [13]. A PDEVS atomic model is based on continuous time, inputs, outputs, states and functions [16]. More complex models are built by connecting several atomic models in a hierarchical fashion. The interactions are ensured by the input and output ports of the models, which favors modularity [17]. A computing grid is a set of interconnected nodes with the same goal. To model a computer grid, we therefore start by proposing the model of a node as an atomic model. Then we propose a grid model as being a coupled model which interconnects atomic models.

### 2.2.1. Model Description with Formal Specifications

The following model presents the formal specifications of an atomic model with PDEVS [18], which is an extension of the DEVS formalism [19]

$$MA = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

With:

- **the set of input ports and values**:

$$X = \{(p,v) \mid p \in Pe, v \in Vx\}$$

where $Pe$ and $Vx$ are two finite sets representing the set of ports and input values;

- **the set of output ports and values**:

$$Y = \{(p,v) \mid p \in Ps, v \in Vy\}$$

where $Ps$ and $Vy$ are two finite sets representing the set of output ports and the values carried by the events generated at the output;

- **the set of sequential states:**

$$S = \{s_i \mid i \in R^+\}$$

- **the internal state transition function**

$$\delta_{int}(S) \to S$$

- **the *external state transition function***

$$\delta_{ext}(Q \times X) \to S$$

where, $X^b$ all input bags belonging to $X$, and $Q$ the **total state set**, $Q = \{(s,e) \mid s \in S, 0 \le e \le ta(s)\}$, $e$ is the **time elapsed** since last transition;

- **the confluent transition function**

$$\delta_{con}(S \times X) \to S$$

- **the output function:**

$$\lambda(S) \to Y$$

- **the time advance function:**

$$ta(S) \to R_0^+ \cup \infty$$

The following model presents the formal specifications of a model coupled with PDEVS

$$MC = \langle X, Y, D, EOC, IC, EIC, Select \rangle$$

With:

- **the set of input ports and values**:

$$X = \{(p,v) \mid p \in Pe, v \in Vx\}$$

where $Pe$ and $Vx$ are two finite sets representing the set of ports and input values;

- **the set of output ports and values**:

$$Y = \{(p,v) \mid p \in Ps, v \in Vy\}$$

where $Ps$ and $Vy$ are two finite sets representing the set of output ports and the values carried by the events generated at the output;

- **the set of the component names:**

$$D = \left\{ mi \mid i \in R^{+} \right\}$$

- **external output couplings connect component outputs to external Outputs:**

$$EOC = \left\{ \left( (mi, psj), (MC, psk) \right) \mid i, j, s, k \in R^{+} \right\}$$

- **internal couplings connect component outputs to component inputs:**

$$IC = \left\{ \left( (mi, psj), (mk, pel) \right) \mid i, j, s, k \in R^{+} \right\}$$

- **external input couplings connect external inputs to component inputs**

$$EIC = \left\{ \left( (MC, pei), (mj, pek) \right) \mid e, i, j, k \in R^{+} \right\}$$

- **The selection function** allowing to resolve the model activation conflict:

$$Select(D) \rightarrow mi[i]$$

### 2.2.2. Graphic Model

The model shown in Figure 1 shows a description of the graphical model with PDEVS.

$$MA = \left\langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \right\rangle$$

## 2.3. Simulation

For the simulations, we use the devs-suite simulator version 4.0.0 developed by the arizona center for modeling and integrative simulation using DEVSJAVA as a language [20]. To simulate the model below, 100 nodes were used to model a calculation grid with the parameters listed in Table 1:

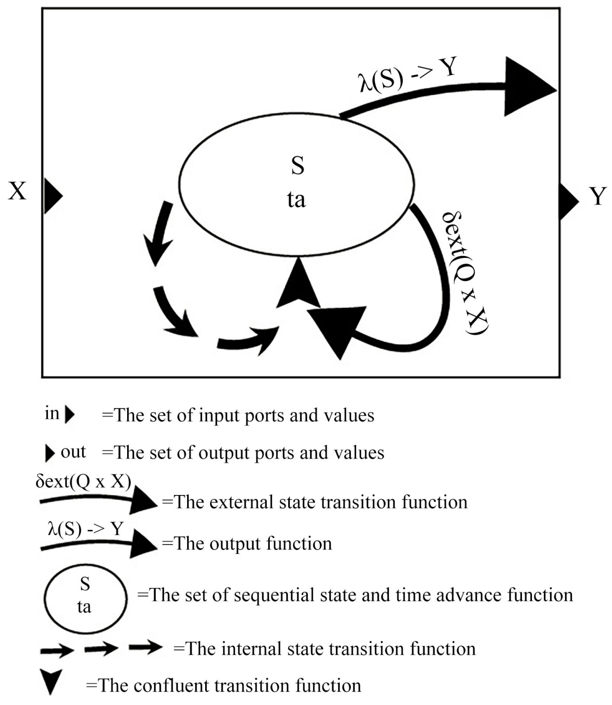

λ(S) -> Y
X
S
ta
Y
δext(Q x X)
in
=The set of input ports and values
out
=The set of output ports and values
δext(Q x X)
=The external state transition function
λ(S) -> Y
=The output function
S
ta
=The set of sequential state and time advance function
=The internal state transition function
=The confluent transition function

Figure 1. Graphical model with PDEVS.

Table 1. Characteristics of the nodes used.

| *Node* | *CPU* | *Cores* | *Tcase* | *RAM size* |
|---|---|---|---|---|
| 1 | 2 × Intel Xeon Gold 6130 | 16 cores/CPU | 87˚C | 192 GiB |
| 3 | 2 × Intel Core i5-3320M | 4 cores/CPU | 73˚C | 12GiB |
| 1 | 2 × POWER8NVL 1.0 | 10 cores/CPU | 74˚C | 128 GiB |
| 1 | 2 × Intel Xeon Gold 5218 | 16 cores/CPU | 87˚C | 1.5 TiB |
| 3 | Intel Xeon Gold 6130 | 16 cores/CPU | 87˚C | 768 GiB |
| 1 | 2 × Intel Xeon E5-2630 v4 | 10 cores/CPU | 74˚C | 256 GiB |
| 2 | 2 × AMD EPYC 7301 | 16 cores/CPU | 65˚C | 128 GiB |
| 4 | 2 × Intel Xeon E5-2680 v4 | 14 cores/CPU | 86˚C | 768 GiB |
| 1 | 4 × Intel Xeon Gold 6126 | 12 cores/CPU | 86˚C | 192 GiB |
| 3 | 2 × Intel Xeon E5-2630L | 6 cores/CPU | 69.8˚C | 32 GiB |
| 1 | 2 × Intel Xeon E5-2698 v4 | 20 cores/CPU | 90˚C | 512 GiB |
| 5 | Intel Xeon E5-2620 | 6 cores/CPU | 77.4˚C | 32 GiB |
| 5 | AMD EPYC 7642 | 48 cores/CPU | 66˚C | 512 GiB |
| 1 | 2 × Intel Xeon E5-2620 v4 | 8 cores/CPU | 74˚C | 64 GiB |
| 2 | 2 × Intel Xeon E5-2630 | 6 cores/CPU | 77.4˚C | 32 GiB |
| 6 | ThunderX2 99xx | 32 cores/CPU | 74˚C | 256 GiB |
| 1 | 2 × AMD Opteron 250 | 1 core/CPU | 65˚C | 2 GiB |
| 1 | 2 × Intel Xeon E5-2630 | 6 cores/CPU | 77.4˚C | 32 GiB |
| 1 | 2 × Intel Xeon Silver 4110 | 8 cores/CPU | 77˚C | 128 GiB |
| 1 | 8 × Intel Xeon E5-2630 v3 | 8 cores/CPU | 72.1˚C | 128 GiB |
| 1 | 2 × Intel Xeon E5-2620 v3 | 6 cores/CPU | 72.6˚C | 64 GiB |
| 1 | 2 × Intel Xeon E5-2650 | 8 cores/CPU | 77.4˚C | 256 GiB |
| 10 | Intel Xeon Gold 5218R | 20 cores/CPU | 87˚C | 96 GiB |
| 1 | 2 × Intel Xeon E5-2650 | 8 cores/CPU | 77.4˚C | 64 GiB |
| 2 | Intel Xeon E5-2650 v4 | 12 cores/CPU | 80˚C | 128 GiB |
| 1 | 2 × Intel Xeon E5-2603 v3 | 6 cores/CPU | 72.8˚C | 64 GiB |
| 2 | Intel Xeon E5-2630 v3 | 8 cores/CPU | 72.1˚C | 128 GiB |
| 6 | Intel Xeon E5-2630 v3 | 8 cores/CPU | 72.1˚C | 128 GiB |
| 1 | Intel Xeon Gold 5220 | 18 cores/CPU | 87˚C | 96 GiB |
| 4 | 2 × AMD EPYC 7452 | 32 cores/CPU | 65˚C | 128 GiB |
| 5 | AMD EPYC 7351 | 16 cores/CPU | 66˚C | 128 GiB |
| 1 | 2 × Intel Xeon Gold 6130 | 16 cores/CPU | 87˚C | 192 GiB |
| 1 | 2 × Intel Xeon E5-2660 | 8 cores/CPU | 73˚C | 64 GiB |
| 7 | 2 × Intel Xeon E5-2630L v4 | 10 cores/CPU | 62˚C | 128 GiB |

**Continued**

| | | | | |
|---|---|---|---|---|
| 1 | 2 × Intel Xeon E5-2660 v2 | 10 cores/CPU | 75˚C | 128 GiB |
| 1 | 3 × Intel Xeon X5570 | 4 cores/CPU | 75˚C | 24 GiB |
| 1 | 2 × AMD Opteron 6164 HE | 12 cores/CPU | 65˚C | 48 GiB |
| 1 | 2 × Intel Xeon E5-2630 v3 | 8 cores/CPU | 72.1˚C | 128 GiB |
| 8 | Intel Xeon E5-2630 v3 | 8 cores/CPU | 72.1˚C | 128 GiB |
| 1 | 2 × Intel Xeon X5670 | 6 cores/CPU | 81.3˚C | 96 GiB |

The 100 nodes are distributed as in the table above. For this modelization, it is considered that the distribution of tasks is perfectly balanced [16] that is to say:

$$\text{Time}_{\text{total}} = \frac{\text{NbFlops}_{\text{total}}(\text{Noeud1})}{\text{Flops/s}(\text{Noeud1})} = \cdots = \frac{\text{NbFlops}_{\text{total}}(\text{Noeudn})}{\text{Flops/s}(\text{Noeudn})}$$

With: **NbFlops** the number of operations performed on all the processors of a node. **Flops/s** the computing power of a node and ***Time*** the execution time of each node of the grid and the tasks are submitted to the system in an identical time interval.

## 3. Results and Discussion

### 3.1. Modeling a Correct Node

A good node is a node that cannot receive failures. The specifications of such a node are shown in Figure 2.

The model starts in an initial "attente" state. when it receives a job via the "tache" port which is an external transition (continuous arrows), the node goes into an "active" state during which the job is processed. At the end of processing, the processed job is returned by the "resultat" port. If several jobs have been sent to the node, the jobs are processed one after the other. Once a job has been processed, the node continues to process the next job thanks to an internal transition (interrupted arrows). If during the processing of a job, another job is sent by the "tache" port, a confluence function will select which of the jobs will be executed between the internal and external transition. The representation of Figure 3 is a formal description of the model of a correct node with the PDEVS formalism

The model having the above formal specifications of a correct node has been simulated and the results of this simulation are shown in the Figure 4.

The simulation of Figure 4 is structured as follows: graph a represents the time of the last event, that is to say the time of the state being processed. Graph b represents the execution time of the next event. The graph c-1 represents the evolution of the states of the system during the execution of the jobs and the graph c-2 represents the state of the system before the arrival of the jobs. The graphs d and e represent the two input ports "tache" and "faute", when a job is submitted for processing to the grid, it enters by the port "tache" and when a fault is committed it enters by the "faute" Port. The graphs f and g respectively represent the output ports "resultat" and "erreur". When a job is successfully processed, the result is returned through the "resultat" port and when the system encounters a failure, the result is returned through the "erreur" port.

The job that are sent to the node are run for 100 seconds. The first tasks that are sent to the node are the first to be executed. The system boots into an "attente" state. When a transition occurs, the node executes the job for the defined execution time. When processing is complete, the corresponding output port takes the corresponding value. In this node, no failure ever occurs so no matter what port entered, the out port "resultat" will always receive an out value on each transition. In the rest of this work, we model a node that can receive failures by omission.

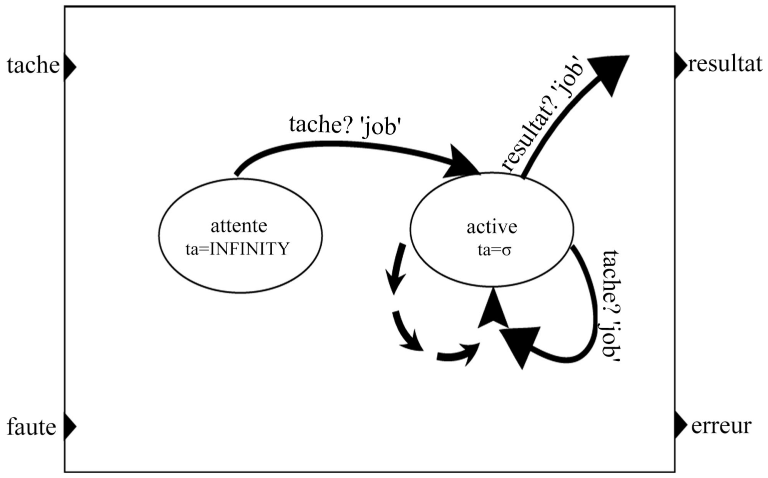


**Figure 2.** Model of a correct node.

$$Noeud = (X_{Noeud}, Y_{Noeud}, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

*Where*

$\mathrm{X}_{noeud} = \{(p,v) \mid p \in IPorts, v \in Xp\}$

$IPort = \{\text{"}tache\text{"}, \text{"}faute\text{"}\}, avec\ \mathrm{Xtache} = \mathrm{Xfaute} = \mathrm{job}$

$\mathrm{Y}_{noeud} = \{(p,v) \mid p \in Oports, v \in Yp\}$

$\mathrm{OPort} = \{\text{"}resultat\text{"}, \text{"}erreur\text{"}\}, \mathrm{avec\ Yresultat = Yerreur = job}$

$\mathrm{S} = \{\text{"}attente\text{"}, \text{"}active\text{"}\}$

$\delta_{ext} = (Q\ \mathrm{x\ tache}):$

- $\circ$ if S = "active" or S = "attente" then S = "active"

$\delta_{int}(S):$

- $\circ$ if S = "active" then S = "active"

$\delta_{con}(Q\ \mathrm{x}\ \mathrm{X}_{Noeud}):$

- $\circ\ \delta_{int}(S)$
- $\circ\ \delta_{ext} = (Q\ \mathrm{x\ tache})$

$\lambda(S) = (resultat, job)$ if S = "active"

$\varnothing$ else

$ta(S):$

- $\circ\ \sigma$ if S = "active"
- $\circ$ INFINITY if S = "attente"

**Figure 3.** Formal description of a correct node.

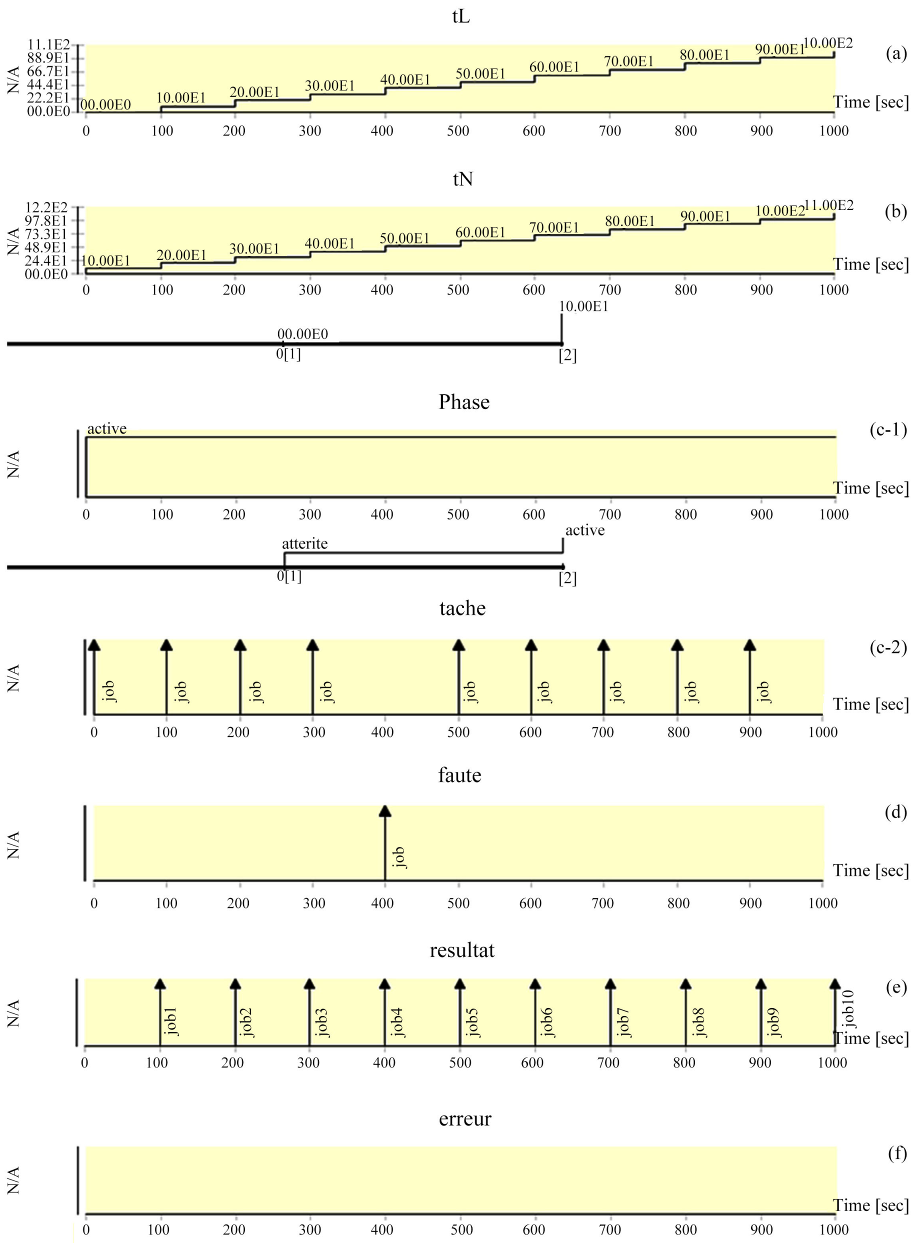


Figure 4. Simulation of a correct node.

## 3.2. Modeling a Failed Node

A node may receive failures by omission from time to time. The specifications of such a node capable of receiving failures by omission are shown in Figure 5 and Figure 6.

A failed node is a good node until a failure occurs, in which case a failure by omission. in this model, the node is in an "active" state until an omission failure occurs and the node enters an omission state. The node remains for a random period which can range from less than a second to infinity. After this period during which the node has remained in an "omission" state, it resumes its normal service and passes to the "active" state. The simulation of Figure 7 represents that of a model failing by omission.

Jobs sent to the node run for 100 seconds. The first jobs sent to the node are the first to be executed. The system starts up in an initial "attente" state. When a transition occurs, the node executes the task for the defined execution time. When processing is complete, the corresponding output port takes on the corresponding value. In this node, when failure by omission occurs in the node (red band), either by overheating of the processor or by overload of the RAM, the node goes into an "omission" state for a random period. In this simulation the failure by omission lasts 90 seconds. After this period, the node resumed its activity as it should. Task 4 will therefore run for 190 seconds and there is now a 90 second lag on all subsequent processed jobs.

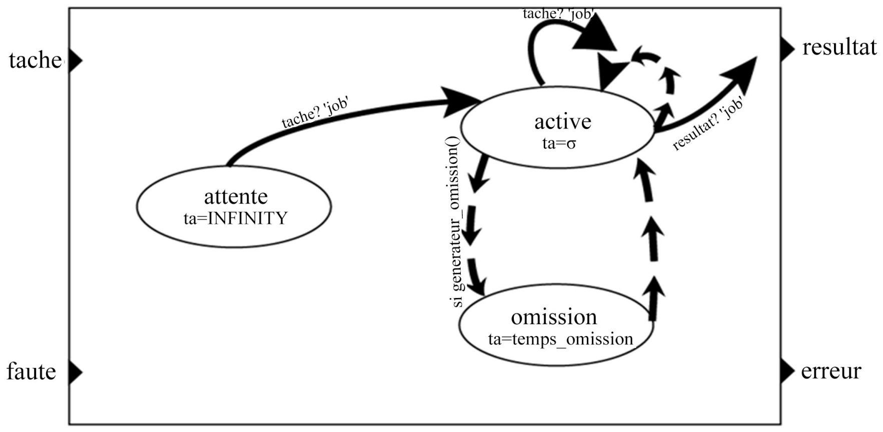


**Figure 5.** Model of a node that can receive failures by omission.

$PDEVS = (X_{Noeud}, Y_{Noeud}, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$

*où*

$X_{noeud} = \{(p,v) \mid p \in IPorts, v \in Xp\}$

$IPort = \{"tache", "faute"\}$, *avec* Xtache = Xfaute = job

$Y_{noeud} = \{(p,v) \mid p \in Oports, v \in Yp\}$

OPort = {"*resultat*", "*erreur*"}, avec Yresultat = Yerreur = job

S = {"*attente*", "*active*", "*omission*"}

$\delta_{ext} = (Q \times \text{tache})$:

∘ if S = "active" or S = "attente" then S = "active"

$\delta_{int}(S)$:

∘ if *generateur_omission* = *faux* then S = "*active*"

∘ else *S* = "*omission*"

$\delta_{con}(Q \times X_{Noeud})$:

∘ $\delta_{int}(S)$

∘ $\delta_{ext} = (Q \times X_{Noeud})$

$\lambda(S) = (resultat, job)$ if S = "active"

$\varnothing$ else

$ta(S)$:

∘ $\sigma$ if S = "active"

∘ *temps_omission* if S = "*omission*"

∘ *INFINITY* if S = "attente"

**Figure 6.** Formal description of a node failing by omission.

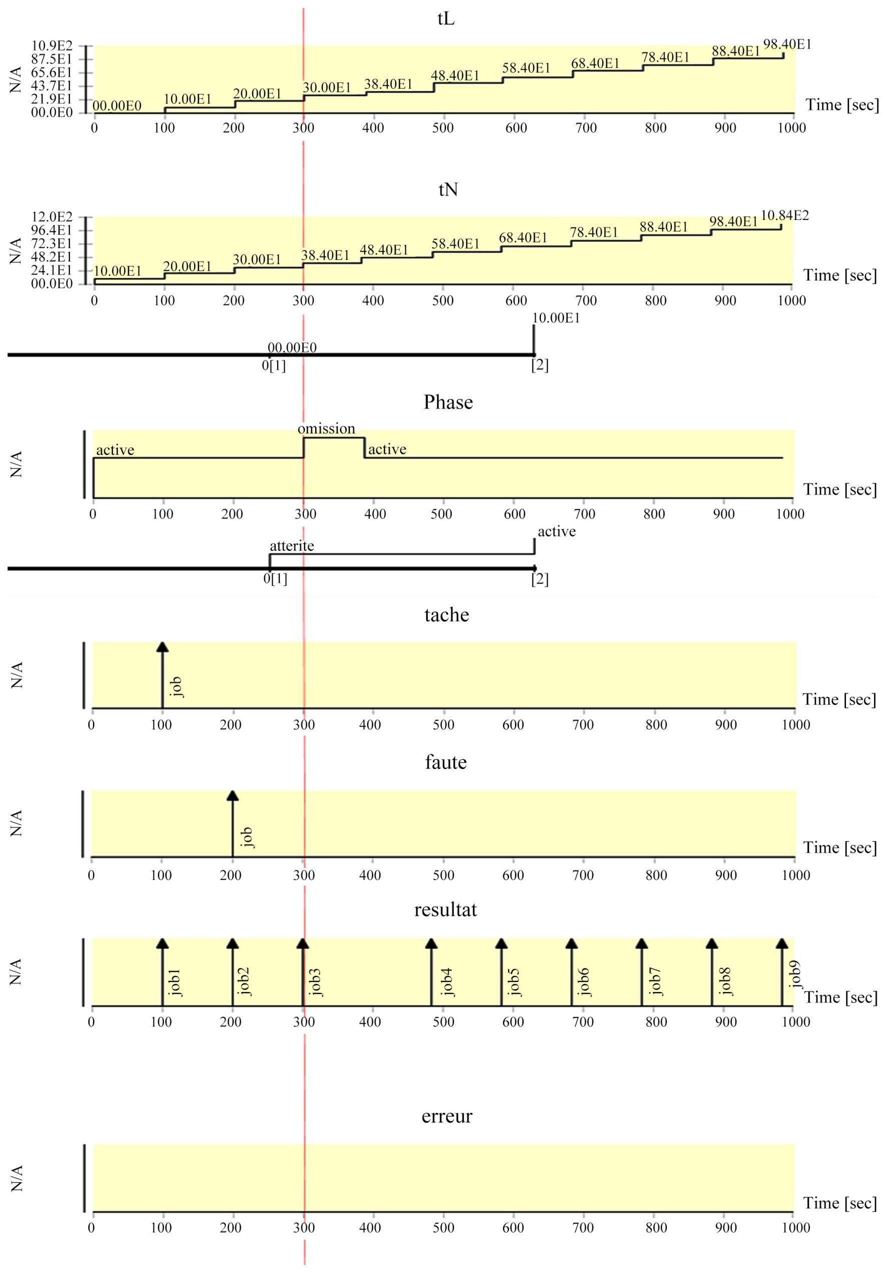


Figure 7. Simulation of a failed node by omission.

## 3.3. Modeling of a Calculation Grid

A computing grid is a set of heterogeneous, interconnected nodes considered as a single computer to provide great computing power. The modeling of a computing grid is therefore, with the PDEVS formalism, a coupled model where the various atomic models which constitute it are the various nodes. The formal specifications of a grid are presented in Figure 8.

In this modelization, the model N represents the calculation grid. Jobs can be submitted through two ports to the grid, "stain" and "fault". The submitted jobs arrive at the "WMSI" task manager, which is responsible for finding available nodes and sending parts of the job to them. Once the jobs have been processed by the various nodes, the parts of the job are sent back to the “WMSO” task manager to be reconstructed, restored and sent back to the output port. The simulation of this model is shown in Figure 9.

In this simulation of a computing grid, every 150 seconds the system makes sure that the nodes are still all connected to the network (green bands) by exchanging messages while performing a system backup. In the event of a failure (red band), a fault tolerance technique is applied to the next message exchange. In the case of our simulation during the processing of job5, a failure occurs. failure detection occurs at t = 450 sec. The fault tolerance technique is applied and the job is processed. In this simulation, the restart recovery caused by node failure is general, that is to say all nodes are restored. Of which at t = 500, the system restarts where the failure occurred.

## 3.4. Proposed Correction Model

In this part we propose a model which makes it possible to anticipate failures by omission in the calculation grid. Unlike the current method of performing system backups at a set frequency and restoring the last backup in the event of a failure, this model looks for signs that can lead to omission failure in every node. The principle is to observe the components of each node using the node’s sensors and exchanged messages. For example, if the temperature sensor of the processor signals a temperature which is approaching due to the temperature of the processor chassis, the failure is reported to the task manager who will no longer send jobs to the node likely to have a failure, the model is therefore represented in Figure 10.

$$N = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC)$$

$$\text{où}$$

$$IPort = \{\text{"tache"}, \text{"faute"}\}, \text{avec Xtache} = \text{Xfaute} = \text{job}$$

$$X = \{(p, v) \mid p \in IPorts, v \in Xp\}$$

$$\text{OPort} = \{\text{"resultat"}, \text{"erreur"}\}, \text{avec Yresultat} = \text{Yerreur} = \text{job}$$

$$Y = \{(p, v) \mid p \in Oports, v \in Yp\}$$

$$D = \{\text{noeud1}, \text{noeud2}, \dots, \text{noeud100}, \text{WMSI}, \text{WMSO}\};;$$

$$M_d = \{M_{\text{noeud1}}, M_{\text{noeud2}}, \dots, M_{\text{noeud}n}, M_{\text{WMSI}}, M_{\text{WMSO}}\};$$

$$\text{EIC} = \begin{Bmatrix} (\text{N}, \text{"tache"}), (\text{WMSI}, \text{"tache"}) \\ (\text{N}, \text{"faute"}), (\text{WMSI}, \text{"faute"}) \end{Bmatrix}$$

$$\text{IC} = \left\{ \begin{matrix} \begin{Bmatrix} (\text{WMSI}, \text{"tache"}), (\text{noeud1}, \text{"tache"}) \\ (\text{WMSI}, \text{"tache"}), (\text{noeud2}, \text{"tache"}) \\ \cdots\cdots\cdots\cdots \\ (\text{WMSI}, \text{"tache"}), (\text{noeud100}, \text{"tache"}) \end{Bmatrix} \\ \begin{Bmatrix} (\text{WMSI}, \text{"faute"}), (\text{noeud1}, \text{"faute"}) \\ (\text{WMSI}, \text{"faute"}), (\text{noeud2}, \text{"faute"}) \\ \cdots\cdots\cdots\cdots \\ (\text{WMSI}, \text{"faute"}), (\text{noeud100}, \text{"faute"}) \end{Bmatrix} \\ \begin{Bmatrix} (\text{noeud1}, \text{"resultat"}), (\text{WMSO}, resultat) \\ (\text{noeud2}, \text{"resultat"}), (\text{WMSO}, resultat) \\ \cdots\cdots\cdots\cdots \\ (\text{noeud100}, \text{"resultat"}), (\text{WMSO}, resultat) \end{Bmatrix} \\ \begin{Bmatrix} (\text{noeud1}, \text{"erreur"}), (\text{WMSO}, erreur) \\ (\text{noeud2}, \text{"erreur"}), (\text{WMSO}, erreur) \\ \cdots\cdots\cdots\cdots \\ (\text{noeud100}, \text{"erreur"}), (\text{WMSO}, erreur) \end{Bmatrix} \end{matrix} \right\}$$

$$EOC = \begin{Bmatrix} (\text{WMSO}, resultat), (N, resultat) \\ (\text{WMSO}, erreur), (N, erreur) \end{Bmatrix}$$

Figure 8. Formal description of the calculation grid.

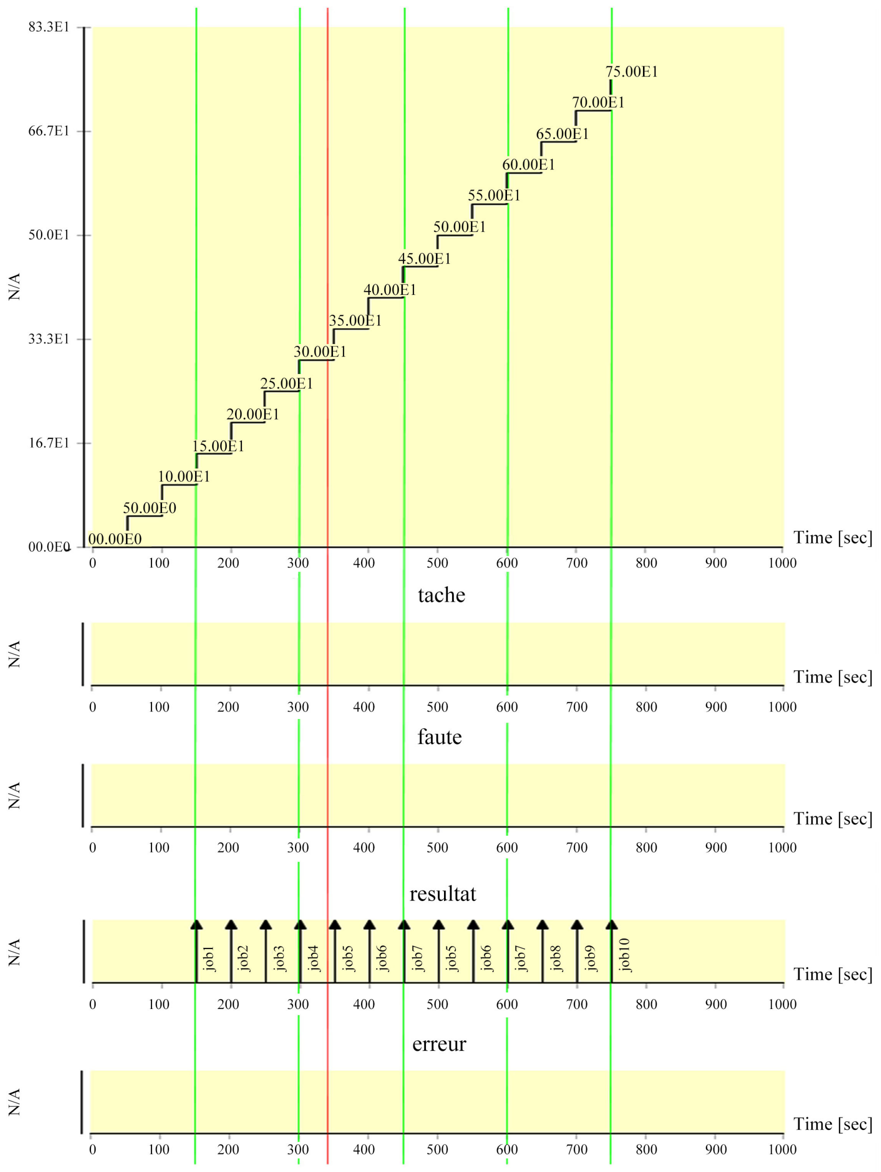


**Figure 9.** Simulation of fault tolerances by omission in calculation grids.

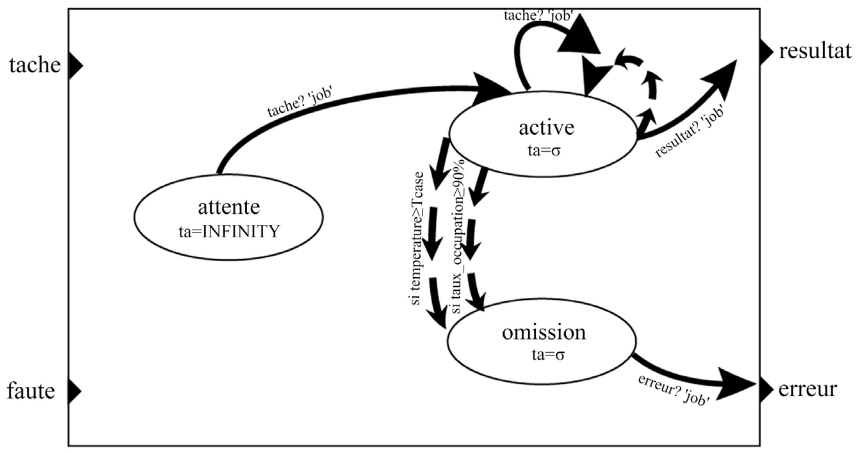
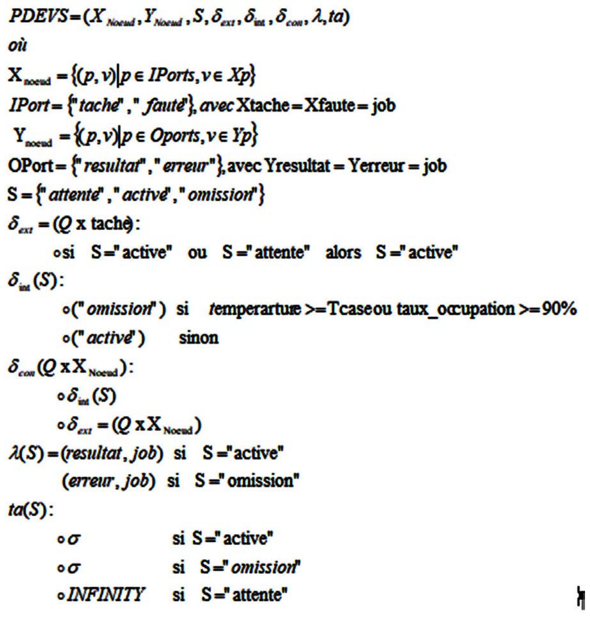


**Figure 10.** Failure-by-omission anticipation model.

The formal description of this model is presented in Figure 11.

The above model presents a model which allows failures by omission to be anticipated. Once the node begins to receive jobs, it goes into the "active" state. During job processing, if the node begins to receive too many jobs beyond what it can handle, the RAM occupancy rate increases to saturation causing failure by omission. Or if, during job processing, the processor temperature rises to approach the temperature of the processor chassis, the node temporarily stops operating indefinitely and causes an omission failure. These values being measured continuously, the principle is to send information on the temperature of the processor and the rate of occupation of the RAM to the task manager which will determine whether it should always forward jobs to this node or not based on the state of RAM and processor. In general, a safe range during normal workload is between 40˚C and 65˚C (or 104˚F - 149˚F) [17]. The Intel Xeon E5-2630L v4 processor has a chassis temperature of Tcase = 62˚C. In this model, if the processor temperature reaches 62˚C, the node reports an "omitted state". Or if the RAM occupancy rate reaches 90% of the usable RAM, the node reports the failure and the grid task manager will no longer send tasks to the node (Figure 12).

This simulation is that of a calculation grid which makes it possible to anticipate failures by omission. In this simulation, three nodes are likely to be bad and the rest are good nodes. If one of the nodes goes down, its processor temperature rises 15 degrees after each task. In this simulation, if the temperature of the Tcase processor chassis minus the T processor temperature is less than 10 degrees (Tcase-T ≤ 10) then a failure is reported in the node and the message is transmitted to the task manager which will ignore thereafter. Here this case occurs after the processing of job4, a node is reported as bad and the task manager



$PDEVS = (X_{Noeud}, Y_{Noeud}, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$

*où*

$X_{noeud} = \{(p, v) | p \in IPorts, v \in Xp\}$

$IPort = \{\text{"tache"}, \text{"faute"}\}$, *avec* Xtache = Xfaute = job

$Y_{noeud} = \{(p, v) | p \in Oports, v \in Yp\}$

OPort = {"*resultat*", "*erreur*"}, avec Yresultat = Yerreur = job

S = {"*attente*", "*active*", "*omission*"}

$\delta_{ext} = (Q \text{ x tache})$:

∘si S="active" ou S="attente" alors S="active"

$\delta_{int}(S)$:

∘("*omission*") si *temperartue* >=Tcaseou taux_occupation >=90%

∘("*active*") sinon

$\delta_{con}(Q \text{ x} X_{Noeud})$:

$\circ \delta_{int}(S)$

$\circ \delta_{ext} = (Q \text{ x} X_{Noeud})$

$\lambda(S)$ = (*resultat*, *job*) si S="active"

(*erreur*, *job*) si S="omission"

*ta*(*S*):

$\circ \sigma$ si S="active"

$\circ \sigma$ si S="*omission*"

∘*INFINITY* si S="attente"

Figure 11. Formal description of the failure anticipation model by omission.

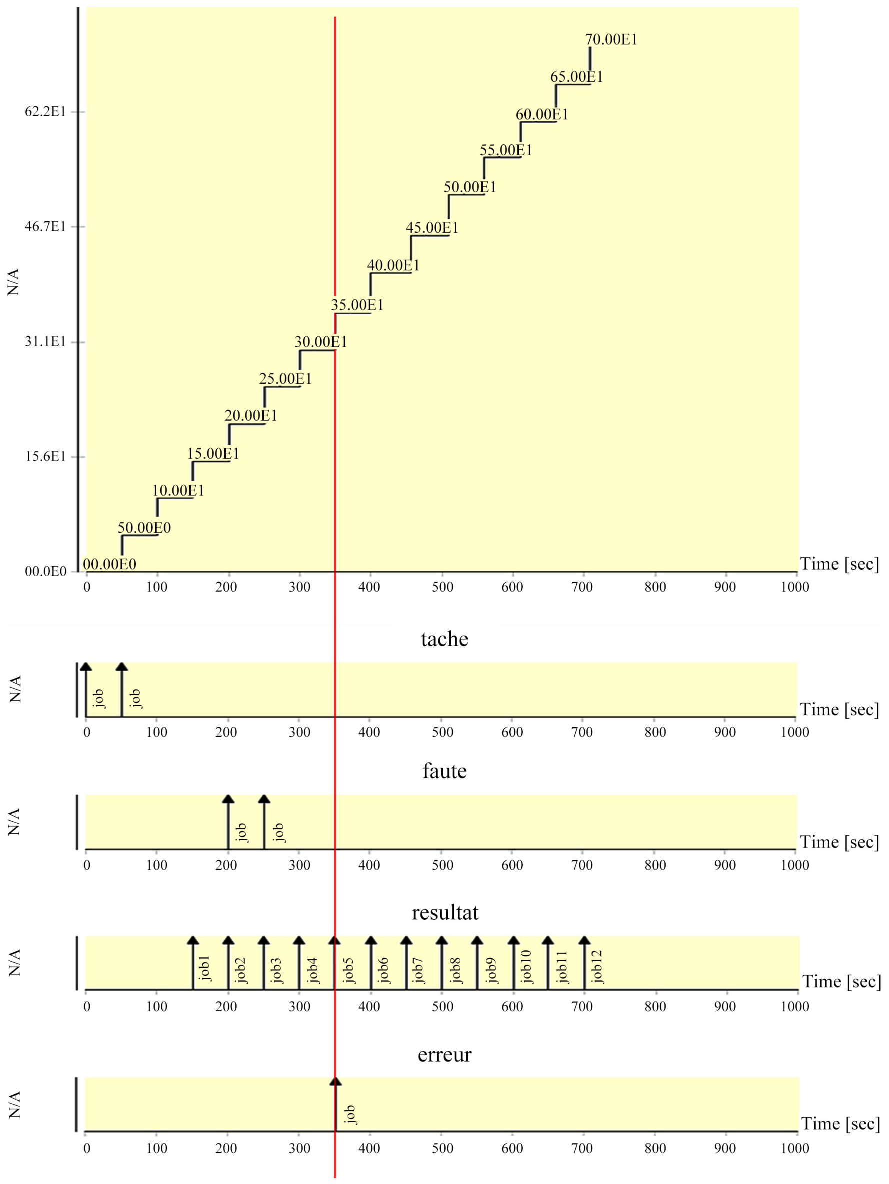


**Figure 12.** Simulation of a calculation grid that can anticipate failures by omission.

ignores it and the system continues to execute the jobs. Unlike other fault tolerance protocols which wait for the failure to occur before tolerating it, this model allows you to anticipate the failure by probing the various components of the node and determining if there may be a failure. This detection time and recovery of the system are almost zero with this model. But this model is only effective if the failure is normal, ie failures which are not linked to the attacks.

## 4. Conclusions

Failures by omission in computational grids have been studied numerically in the present work. The numerical methodology is based on the PDEVS modeling formalism in the devs-suite simulator version 4.0.0. The main results are summarized as follows:

1) We first present a correct node model which is a node which cannot receive failures and a node model which can receive failures by omission.

2) Next, we present a computational grid model which is a set of interconnected nodes. This grid which is a set of correct nodes and nodes that can receive failures by omission.

3) Finally, we propose a model which makes it possible to anticipate failures by omission in the calculation grids. This model is based on the behavior of certain components of the node to anticipate failures. Using this model will help limit omission failures that are caused by the hardware components of the node.

## Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

## References

[1] Kalfadj, F., Belabbas, Y. and Meddeber, M. (2009) Conception d'un Simulateur de Grilles Orienté Gestion d'Équilibrage. *Proceedings of the* 2*nd Conférence Internationale sur l'Informatique et ses Applications*, Saida, 3-4 May 2009, 1-11.

[2] Barchet-Estefanel, L.A. (2005) LaPIe: Communications Collectives adaptées aux Grilles de Calcul. Institut National Polytechnique De Grenoble, Grenoble.

[3] Taïani, F., Killijian, M.O. and Fabre, C.J. (2006) Intergiciels pour la tolérance aux fautes. *RSTI-TSI*, **25**, 599-630. https://doi.org/10.3166/tsi.25.599-630

[4] Monnet, S. (2006) Gestion des données dans les grilles de calcul: Support pour la tolérance aux fautes et la cohérence des données. Université Rennes 1, Rennes.

[5] Rebbah, M. (2015) Tolérance aux fautes dans les grilles de calcul. Université des Sciences et de la Technologie d'Oran, Oran.

[6] Ndiaye, N.M. (2013) Techniques de gestion des défaillances dans les grilles informatiques tolérantes aux fautes. Université Pierre et Marie Curie, Paris.

[7] Akoka, J. and Comyn-Wattiau, I. (2006) Encyclopédie de l'informatique et des systèmes d'information. Vuibert, Paris.

[8] Krakoviak, S. (2004) Tolerences aux fautes, Université Joseph Fourier.

[9] Ramat, É., Duboz, R. and Quesnel, G. (2012) Theorie de la modélisation et de la simulation, fondements formels et operationnels de record/VLE, INRA-Cirad-ULCO/LISIC.

[10] Zeigler, B.P., Kim, T.G. and Praehofer, H. (2000) Theory of Modeling and Simulation. Academic Press, Orlando.

[11] Foster, I. and Kesselman, C. (2003) The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, Burlington.

[12] Avizienis, A., Laprie, J.-C. and Randell, B. (2001) Fundamental Concepts of Dependability. Newcastle University, Newcastle.

[13] Zeigler, B., Gon Kim, T. and Praehofer, H. (2000) Theory of Modeling and Simulation. 2nd Edition, Academic Press, New York.

[14] Perez, J. (2009) Apprentissage Artificiel pour l'ordonnancement des tâches dans les grilles de calcul. Université Paris-Sud, Paris.

[15] Zeigler, B.P. (2005) Introduction to DEVS Modeling and Simulation with JAVA:

Developing Component-Based Simulation Models. Arizona Center for Integrative Modeling and Simulation, Tucson.

[16] Kadri, W. (2012) Placement dynamique des tâches dépendantes dans une grille de calcul. Université d'oran, Oran.

[17] Villinger, S. (2021) Comment surveiller la température de votre processeur. https://www.avast.com/fr-fr/c-how-to-check-cpu-temperature

[18] Ouedraogo, D., Wendsida Igo, S., Sawadogo, G.L., Compaore, A., Zeghmati, B. and Chesneau, X. (2020) Modeling and Numerical Simulation of Heat Transfers in a Metallic Pressure Cooker Isolated with Kapok Wool. *Modeling and Numerical Simulation of Material Science*, **10**, 15-30. https://doi.org/10.4236/mnsms.2020.102002

[19] Guessoum, Z., Briot, J.P., Faci, N. and Marin, O. (2004) Un mécanisme de réplication adaptative pour des SMA tolérants aux pannes. JFSMA.

[20] Zeigler, B. (1976) Theory of Modeling and Simulation. Academic Press, London.