Activation and Termination Detection on Distributed Systems

Waldemar Baraldi, Diego Scarpa, Ignacio Ponzoni and Jorge Ardenghi

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Av. Alem 1253 - CP 8000 - Bahía Blanca - ARGENTINA

Abstract

Facing the different approaches to process activation and global termination detection on distributed systems, this paper performs a practical comparison between the mainly opposed centralized and distributed models. Firstly, they were implemented to solve both the activation and termination detection problems. Based on processing performance and communication overhead metrics, both systems were evaluated running on various parametrized environments. As a result, we present the values obtained and the final conclusions.

The overall design was made following an object oriented methodology, which was coded using *C++*. The inter-process communication was carried out using the *PVM* libraries.

1 Introduction

In the field of parallel processing,the method used to descompose a large problem into several concurrent small tasks, has emerged as a key enabling technology in modern computing. The past decade has witnessed an ever-increasing acceptance and adoption of parallel processing, both for high-performance scientific computing and for more "general-purpose" aplications. This was the result of a demand for higher performance, lower cost, and sustained productivity. The acceptance has been facilitated by two major developments: massively parallel processors (MMPs) and the widespread use of distributed computing [1].

The basic idea behind parallel programming is the division of a problem into several smaller tasks which can be solved simultaneously. At this point many distributed algorithms and load balancing schemes emerge to take advantage of the parallel processing. The need for a global termination detection strategy therefore arises due to the use of problem division. When a computation is distributed, recognizing that it has come to an end may be difficult unless the problem is such that one process reaches the solution. In general, distributed termination at time t requires the following conditions to be satisfied [2]:

- Application-specific local termination condition exist throughout the collection of processes, at time t.
- There are no messages in transit between processes at time t.

The objective of this paper is to find two implementations such that they can be compared from a practical as well as a theorical point of view. The different environments used for the evaluation of these implementations were chosen purposing to reproduce distributed computing real life problems. It is also important to note that the scenarios where only one process reaches the solution were considered trivial and therefore discarded.

On the contrary, we focused on modelling systems where processes were capable of activating children nodes which possibly can survive their creator. In consequence, global termination happens when every process decides to finish its execution by itself.

The children nodes activation method, *i.e.* the method used to choose among candidate nodes (idle at the time), is a very important one, as is the information needed to make this choice. Once again, this problem allows either a centralized or a distributed solution. In order to keep the coherence of both implementations and not affect the results, the centralized termination model uses a centralized children activation method, and the distributed one uses its respective ring-based fashion method.

2 Design

Three different entities were clearly set apart in the design of both systems: the *Worker* which is the process that carries out the solution of tasks

inside a node, and the *Local Termination Manager (LTM)* which, combined with the *Global Termination Manager (GTM)*, performs the necessary cleaning to make the activation and termination detection algorithms work properly.

Both system are composed of many worker nodes subordinated to a unique manager node. The worker nodes solve the problem, which involves deciding when to be *idle* and when a new worker node must be set *active* . The manager node coordinates all these activities according to the implemented model.

Each worker node contains one *Worker* instance and one *LTM* instance which run on different threads. This allows the former to work on the task while the latter waits for incoming messages. These nodes exist throughout the whole execution of the system alternating between the *active* and *idle* states. When a worker node is in the *idle* state, its *Worker* instance is latent, waiting for the *LTM* instance to receive an activation signal. Once this signal arrives the *Worker* switches to the *active* state and starts consuming CPU cycles. From time to time it stops and decides whether to be *idle* or to activate a node. In any case the *Worker* communicates the requirement to the *LTM* which in turn becomes responsible for its fulfillment.

Notice that once the *Worker* asks for the activation of another worker node to the *LTM*, it resumes consuming CPU cycles while the *LTM* negotiates with the global system for an effective node activation. Every activation request issued by a *Worker* activates an *idle* process unless all processes in the system are already in the *active* state. In that case the *LTM* discards the request. The *Worker* nodes are not allowed to switch back to the *idle* state if they have pending requests.

The ultimate goal of the system is the efficient management of the activations and deactivations of worker nodes. In other words, when an activation request is issued the system must rapidly find out whether it can be performed or it has to be discarded. Moreover, when a deactivation occurs it must realize if the termination conditions are satisfied. Recall that communication efficiency is central to the performance of distributed systems.

The environment parameters formerly mentioned define the probability of node activation and deactivation. These two probabilities were included in order to evaluate the implementations' performance in terms of different environments. The first one defines the likelihood that a *Worker* will choose to activate a child when it has the chance (AP, *Activation Probability*). The second one defines the probability of a worker node local termination (TP, *Termination Probability*). For example, an environment where both the activation probability and the local termination probability are high characterizes a system where worker nodes need many children to solve their tasks but single tasks are in fact very simple.

Worker instances do not directly know the probability values because they are hidden behind another object, instance of a class called *ActionGenerator*. This object generates actions with uniform distribution in accordance to the given probabilities. Possible actions are *activate a new worker node*, *terminate* and *continue*.

Just before starting to solve the problem, the configuration phase takes place. As part of it, the *GTM* provides each *Worker* node instance with an *ActionGenerator*. Then, at run time, each *Worker* has its own *ActionGenerator* instance to which it can ask for actions to perform.

In the two following sections we explain the particular facts about both the centralized and distributed approaches.

3 Centralized Model

Due to its simplicity, the first model described is the centralized one. As seen in the previous section, both systems' architectures have one manager node and many worker nodes. In the centralized model, the manager node has complete knowledge of the states of every worker node. Making use of this information, it coordinates every activation request and local termination acknowledge.

As shown in Figure 1, each worker node is mainly composed of two objects which carry out the task related to the solution of the problem (*Worker*) and the coordination with the manager node (*CentLTM*). This last object is an instance of the *CentLTM* class derived from *LTM* that implements its functionality.

Following the centralized strategy, when some *Worker* decides to activate another node, it sends the request to the *CentLTM* which forwards it to the *CentGTM* (*GTM* derived class instance) present at the manager node. Knowing the state of each node, the *CentGTM* object chooses one worker node among the *idle* ones and switches it to the *active* state through its local *C*entLTM object. Once again, if the system is on its maximum load, none is activated, the request is discarded, and the asking *CentLTM* notified.

On the other hand, when a *Worker* decides to switch back to the *idle* state, it notifies its decision to the local *CentLTM* object which forwards it to the *CentLTM*. When the *CentGTM* receives this notification it checks for active worker nodes. If there are none, the *CentGTM* can guarantee that both termination conditions have been satisfied. Finally, one more message is sent to the worker nodes to report the global termination so they can free their local allocated resources.

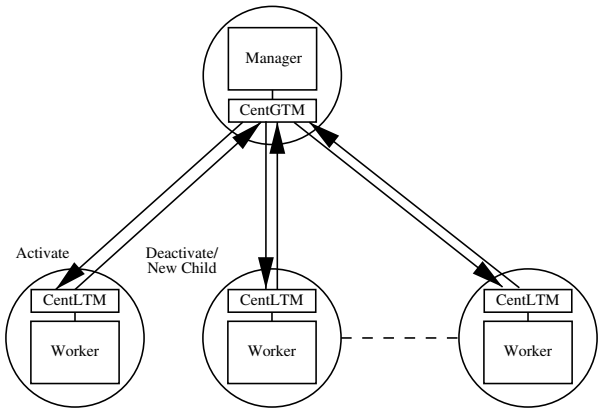

Figure 1: Centralized Termination

4 Distributed Model

Most common distributed detection termination models are based on ring structures. In this paper we implement the dual-pass ring termination algorithm [3] adding two slight variations.

The first one consists of the distributed node activation mechanism explained in Section 1. The second variation adds a new termination token colour to inform each node in the ring that the global termination conditions are satisfied.

Analogous to the centralized version, this model is based on the *RingLTM* and *RingGTM* classes derived from *LTM* and *GTM* respectively. Among the worker nodes, there is a distinguished one called first node in the ring. This node is responsible for changing the termination token colour in each pass.

Tokens come in two flavours depending on their purpose and the information they carry:

Activation Token As explained in Section 2, this model implements a distributed node activation scheme which must run efficiently over a ring structure. A new token type, called activation token, is added. It carries an activation request around the ring looking for an *idle* worker node. Notice that there is one activation token for each request made. In case of reaching back its creator without finding an *idle* worker node, the local *RingLTM* must decide what to do. A second pass can be attempted or the request can just be discarded and the local *Worker* notified. The information carried by this token is its creator *id*.

Termination Token Temination tokens are used for termination detection purposes. Only one token of this type circulates through the ring but it can be of different colours. Its functionality is exactly the same as in [4] but we have added the red colour. The colour meanings are the following:

White: According to the dual-pass termination algorithm the token is white when it has not visited any node which had activated a previous node in the ring.

Black: On the contrary, the token is coloured black when it visits a node which has activated a previous node in the ring. The distinguished node is responsible for turning the token back to white after each pass is completed.

Red: When the first node receives a white token, all nodes are *idle* . In that case it makes a red token circulate through the ring informing every node that the global termination was reached and that they can release their local allocated resources.

Figure 2 shows the distributed model scheme.

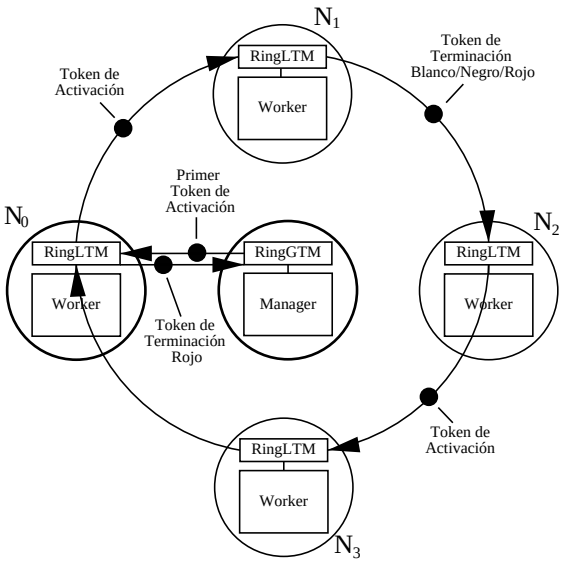

Figure 2: Distributed Termination

5 Performance analysis

5.1 Measures and metrics

Performance metrics were calculated based on three measures taken from each execution:

Number of activations (NA) Total number of succesfully activated nodes.

Number of discarded requests (ND) Total number of activation requests that did not activate a worker node. Useful for evaluating the activation method.

Execution Elapsed Time (EET) Elapsed time from the moment that the first activation request is issued until the global termination is reached, expressed in seconds.

According to these measures, the following metrics were calculated:

Number of activation requests (NR) Total number of activation requests managed by the system. Obtained as $NR = NA + ND$.

Requests per Second ($R_{\times s}$) Number of requests that the system managed per second of execution. This metric is useful in comparing activation efficiency on systems with the same activation probability. Calculated as follows:

$$R_{\times s} = \frac{NR}{EET}$$

Activations per second ($A_{\times s}$) Number of activations yielded per second of execution. This metric is useful in calculating activation efficiency on finding *idle* nodes. Obtained as follows:

$$A_{\times s} = \frac{NA}{EET}$$

Request discard ratio (DR) Discarded activation request ratio. The activation method is supposed to fulfill as many activation requests as possible. This metric evaluates to which degree it has succeded.

From now on, the acronym *Avg.* means the average of values found on the executions with the same input parameters. Besides, *Var* means the variance calculated from those values.

5.2 Execution environments

Every execution was carried out on Pentium 200Mhz. computers connected with a 10Mbit Ethernet network. Each case study was evaluated performing 10 executions with the same input parameters. The executions with a number of activations smaller than the number of nodes in the system were laid apart. The last case is an exception where exactly the opposite was meant.

Due to the Ethernet interconnection nature, no physical communication parallelism is possible. In order to provide some communication parallelism more than one node per computer is run (affecting CPU overhead, of course). The notation is the following:

$$W = computers \times nodes\_per\_computer$$

For instance, $W = 5 \times 2$ means that there are 10 worker nodes, 5 computers with 2 nodes each.

Messages between nodes in the same computer do not make use of the physical interconnection. Thus, more than one message can be sent at the same time.

5.3 Case Study 1

In this case a system with a significant amount of computation and one node per computer is represented. Then, all the communication takes place through the physical interconnection network, disallowing any communication parallelism. As shown in Figure 3, $AP = 0.05$ and $TP = 0.10$ with 5 computers running only one process each.

The results are as follow:

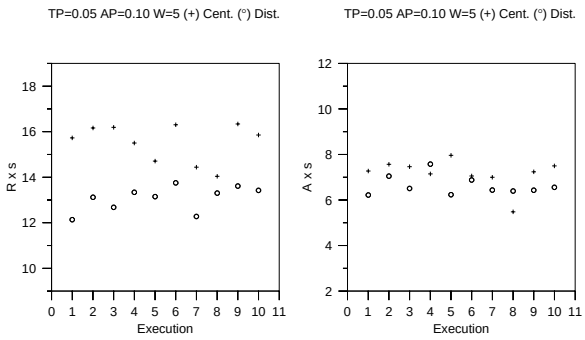

Figure 3: $R_{\times s}$ and $A_{\times s}$ in study case 1

Impl.	$Avg.R_{\times s}$	$Var.R_{\times s}$	$Avg.A_{\times s}$	$Var.A_{\times s}$	DR
Cent.	15.52	0.63	7.16	0.38	53.83
Dist.	12.88	0.90	6.40	0.68	50.45

The centralized implementation clearly overcomes the distributed one. It shows better general performance and stability but it suffers a fairly high request discard ratio.

5.4 Case Study 2

This case keeps the previous input parameters but duplicates the number of worker nodes, placing two of them in each computer. This highly increases CPU utilization. When communication between nodes in the same computer takes place, some communication parallelism is achieved because they do not make use of the physical network. Notice that it is only needed in the distributed approach, so it will benefit the ring based implementation without affecting the centralized one.

The results are as follow (See also Figure 4):

Impl.	$Avg.R_{\times s}$	$Var.R_{\times s}$	$Avg.A_{\times s}$	$Var.A_{\times s}$	DR
Cent.	14.69	1.89	7.61	0.01	47.82
Dist.	10.44	0.02	6.31	0.04	39.53

The communication parallelism benefit does not counteract the high CPU overhead caused by

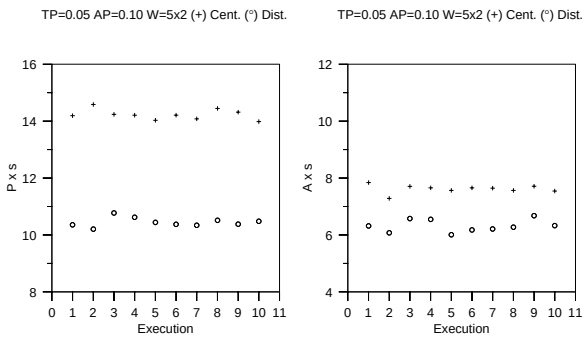

Figure 4: $R_{\times s}$ and $A_{\times s}$ in study case 2

such activation probability. The increase in communication between adjaeent nodes negatively influences the distributed implementation. Much of that communication is carried out by the CPU and therefore its performance falls significantly. In spite of this low performance it has both low variance and low request discard ratio.

5.5 Case Study 3

A system with high CPU usage but low task generation is characterized in this third case (Figure 5). Much communication is needed and it is carried out through the physical interconnection network.

The results are as follow:

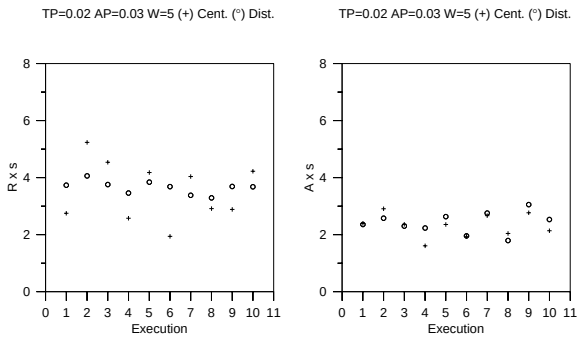

Figure 5: $R_{\times s}$ and $A_{\times s}$ in study case 3

Impl.	$Avg.R_{\times s}$	$Var.R_{\times s}$	$Avg.A_{\times s}$	$Var.A_{\times s}$	DR
Cent.	3.31	1.18	2.17	0.22	30.43
Dist.	3.65	0.04	2.41	0.12	33.81

The distributed model fairly beats the centralized one in terms of managed requests and activations per second, obtaining low variance but high request discard ratio.

5.6 Case Study 4

The activation and termination probabilities are increased keeping the same proportion. The number of nodes per computer is duplicated. See Figure 6.

The results are as follow:

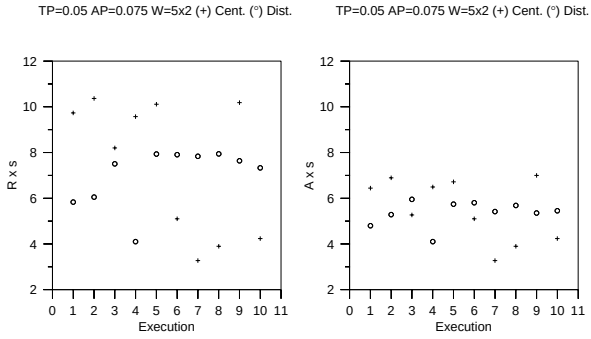

Figure 6: $R_{\times s}$ and $A_{\times s}$ in study case 4

Impl.	$Avg.R_{\times s}$	$Var.R_{\times s}$	$Avg.A_{\times s}$	$Var.A_{\times s}$	DR
Cent.	7.44	7.84	5.51	1.65	20.06
Dist.	7.00	1.47	5.35	0.27	22.00

This case is rather similar to Case Study 2; however, the lower activation probability and CPU usage allows the distributed model to take advantage of the communication parallelism. Thus, it improves considerably, almost reaching the performance of the centralized model. The distributed model request discard ratio is higher and its variance lower.

5.7 Case Study 5

In this case only 2 computers are used and 5 nodes are placed on each one. The idea is to test CPU overhead vs. communication paralellism.

The results are as follow:

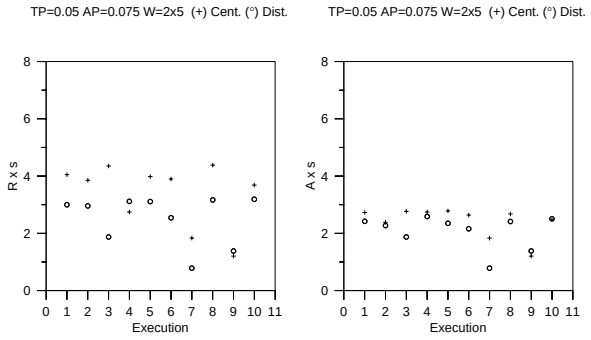

Figure 7: $R_{\times s}$ and $A_{\times s}$ in study case 5

Impl.	$Avg.R_{\times s}$	$Var.R_{\times s}$	$Avg.A_{\times s}$	$Var.A_{\times s}$	DR
Cent.	3.39	1.08	2.42	0.23	24.12
Dist.	2.51	0.67	2.07	0.29	14.39

As seen in Case Study 2 the CPU overhead caused by the internal messages is significantly higher than the benefit given by communication parallelism. Therefore, the distributed implementation performance falls considerably. On the other hand, the centralized model yields a higher request discard ratio and lower variance.

5.8 Case Study 6

Finally both implementations are compared by the extreme case of $AP = 1$ and $TP = 0.5$. Only one node is placed on each computer so messages go through the physical inteconnection network. See Figure 8.

The results are as follow:

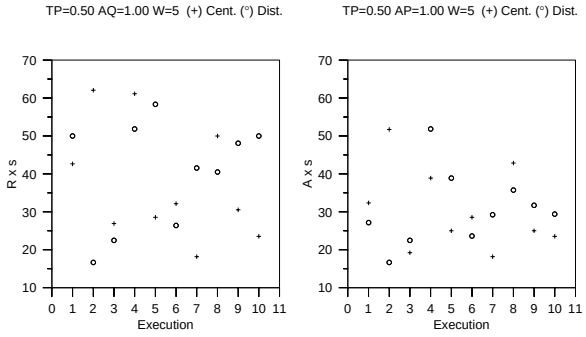

Figure 8: $R_{\times s}$ and $A_{\times s}$ in study case 7

Impl.	$Avg.R_{\times s}$	$Var.R_{\times s}$	$Avg.A_{\times s}$	$Var.A_{\times s}$	RD
Cent.	37.46	213.01	30.44	103.20	16.18
Dist.	40.74	183.86	30.78	89.06	20.60

The distributed activation benefits are clearly shown in this case. The centralized implementation needs both activations and deactivations to be managed by only one node. This node, facing such a high activation probability, gets saturated.

6 Conclusion

Two alternatives for activation and termination detection models in distributed systems were presented and implemented in this paper. After evaluating various execution environments we found those which could lead us to a worthwile comparison.

Although the centralized implementation is never considerably overcome by its distributed counterpart, we must take into account that every evaluation was executed over an Ethernet physical network. The ring based approach is significantly damaged by this fact since all the communication is carried out sequentially, sharing only one physical channel.

The fact that must be brought out is that the amount of available communication parallelism is central to the distributed approach performance. Thus, we can assert that the main cause of its lower performance is the single channel network configuration.

On the other hand, we found out that the system which yields better performance suffers a higher ratio of discarded activation requests. This might be considered natural for a system working most of the time at maximum capacity.

Therefore, we can conclude that the centralized model is the best choice for systems with a small number of nodes and a shared single channel communication arquitecture. On the contrary, when the number of nodes grows and the need for communication gets higher, the distributed model becomes a competitive alternative. It is feasible that on a ring based physical communication arquitecture the distributed approach will be the only one to consider.

References

[1] A. L. BEGUELIN, J. J. DONGARRA, G. A. GEIST, W. C. JIANG, R. J. MANCHECK, V. S. SUNDERAM, *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press, Cambridge Massachusetts, London, England.

[2] BERTSEKAS, D. P., AND J. N. TSITSIKLIS (1989), *Parallel and Distributed Computation Numerical Methods*, Prentice Hall, Englewood Cliffs, New Jersey.

[3] WILKINSON, B., AND M. ALLEN (1999) *Parallel Programming : techniques and applications using networked workstations and parallel computers*, Prentice Hall, Upper Sadle River, New Jersey.

[4] E.W. DISJKSTRA, W.H.J. FEIJEN, A.J.M. VAN GASTEREN *Derivation of a Termination Detection Algorithm for Distributed Computations.* Information Processing Letters 16(5), 1983, pp 217-219.