Detección de Terminación y Activación en Sistemas Distribuídos

Waldemar Baraldi, Diego Scarpa, Ignacio Ponzoni y Jorge Ardenghi

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

Resumen

Frente a las alternativas de activación de procesos y de detección de terminación en sistemas distribuídos, este trabajo realiza una comparación práctica de los principales exponentes antagónicos. Se implementaron los modelos centralizado y de anillo, tanto para la activación de procesos como para la detección global de terminación. Luego, basados en métricas que midieran la performance de procesamiento y la sobrecarga de comunicación, se evaluaron ambos métodos variando diferentes parámetros de ejecución. Como resultado se entregan los valores hallados y las conclusiones que podemos inferir.

El diseño general de ambas implementaciones se realizó siguiendo la metodología de orientación a objetos, se codificó en C++ y para la comunicación se utilizaron las bibliotecas PVM.

1. Introducción

El procesamiento paralelo, como se denomina el método que consiste de varias pequeñas tareas que resuelven un único gran problema, emergió en la actualidad como la clave de la computación moderna. Durante los últimos años hemos sido testigos de una creciente adopción del procesamiento paralelo tanto para la computación cientifica de alta performance, como para aplicaciones de propósito general. Esto se debe a la continua necesidad de mayor performance a cambio de un menor costo [1].

La idea básica detrás del procesamiento paralelo es la partición del problema en tareas considerablemente mas pequeñas que puedan resolverse de manera simultánea. Aqui surgen las metodologias y propuestas de algoritmos distribuidos y de balance de carga que buscan explotar al máximo el paralelismo de procesos. Como consecuencia de la partición existe la necesidad de una estrategia mediante la cual sea posible determinar el momento en que el sistema alcanza un punto global de finalización.

Cuando la computación es distribuida, reconocer que se ha alcanzado este punto de terminación puede resultar dificultoso a menos que el problema sea tal que un solo proceso alcance la solución. En general, la terminación distribuída en un momento t requiere que las siguientes condiciones sean satisfechas [2]:

- Todos los procesos han alcanzado sus respectivas condiciones de terminación local en el momento t.
- No hay mensaje en tránsito entre los procesos en el momento t.

Cualquier estrategia o modelo con el cual se enfrente este problema deberá buscar el momento t en que se satisfagan ambas condiciones.

En este trabajo la motivación es encontrar dos implementaciones teórica y prácticamente comparables, que reproduzcan escenarios posibles al momento de resolver problemas reales. Se descartaron aquellos problemas triviales en donde todos los procesos que componen el sistema buscan una misma solución y una vez hallada por cualquiera de ellos el sistema termina. Por el contrario se buscó modelar sistemas en los cuales los procesos en ejecución puedan necesitar activar subprocesos que tengan la posibilidad de sobrevivir a su creador. Y en consecuencia, la terminación del sistema se produzca cuando todos los procesos cambien al estado de inactividad por *motus* propio.

Cabe destacar que al momento del diseño se reconoció la importancia del método de activación de subprocesos, más técnicamente, el método de selección entre los procesos candidatos (inactivos en ese instante) y el mantenimiento de la información pertinente. Fueron factibles, entonces, las posibilidades de realizar la selección de manera tanto centralizada como distribuída. Y aunque en principio esta decisión es independiente del método de detección de terminación, la implementación de un modelo de activación distribuído junto con uno de detección de terminación centralizado (o viceversa) resultaba poco natural y podía impedir una interpretación coherente de los resultados de desempeño. Es por eso que se implementaron dos sistemas, basados en activación y detección de terminación distribuída y centralizada respectivamente.

2. Nociones de la especificación de diseño

En ambos modelos se separaron ostensiblemente las nociones de *Worker*, proceso dedicado a la solución del problema y de Administrador Local de Terminación (LTM[1]), ubicados en cada uno de los nodos y el Administrador Global de Terminación (GTM[2]) ubicado en el nodo manager.

Cada uno de estos objetos tienen la responsabilidad de llevar a cabo las tareas locales en cada nodo para que se realice correctamente tanto la activación de procesos como la detección de terminación global del sistema. En ambos casos estas nuevas dos clases corren en *threads* diferentes dentro de cada nodo. Esto se debe a que en ambos casos es necesaria la ejecución de otros procesos al mismo tiempo y ellos se encontrarán a la espera de mensajes.

Existirán solo dos tipos de nodos, aquellos que se encargan de resolver el problema y de decidir cuando desactivarse o cuando activar un hijo y aquel que se encargará de coordinar todas estas actividades. Los primeros, que desde ahora llamaremos nodos trabajadores, existirán mientras el sistema se encuentre en funcionamiento, pero tendrán dos posibles estados: activo e inactivo. Cuando un nodo está inactivo, su *Worker* se encuentra latente esperando que su *LTM* reciba una señal de activación. Recibida esa señal, el *Worker* cambiará a estado activo y comenzará a consumir ciclos de *CPU*. De tanto en tanto se encontrará en la disyuntiva de continuar consumiendo *CPU*, activar un nuevo proceso o desactivarse. En todos los casos se lo comunicará al *LTM* y este hará lo propio con respecto al sistema global.

Cabe destacar que si el *Worker* desea activar un nuevo proceso, lo comunicará a su *LTM* y luego continuará con el consumo de *CPU* mientras este último se encargará de efectivizar la activación. Toda solicitud de activación por parte del *Worker* tendrá como resultado la activación en sí de un proceso o, en caso de que el sistema se encuentre en su máxima carga (todos los nodos están activos), el descarte de la solicitud. Ningún nodo podrá desactivarse si tiene pendiente resultados de solicitudes de activación.

El objetivo final del sistema es administrar las activaciones y desactivaciones de manera eficiente, es decir, frente a una solicitud de activación responder lo antes posible llevándola a cabo o descartándola y, frente a una desactivación controlar eficazmente si el sistema ha llegado a su punto de terminación. Debemos recordar que en los sistemas distribuídos la eficiencia de comunicación entre las componentes es crucial en el funcionamiento eficiente del sistema como un todo.

Los parámetros mencionados anteriormente definen las probabilidades mediante las cuales el *Worker* decidirá si necesita activar un nuevo proceso o desactivarse localmente. Estas probabilidades fueron incluídas para poder comparar el desempeño de los modelos en diferentes ambientes que pueden presentarse cuando se resuelven problemas reales. La primera es la probabilidad con la que un *Worker* decidirá efectivamente solicitar la activación de un nuevo proceso cada vez que tenga oportunidad. La segunda define con que probabilidad cada *Worker* decidirá desactivarse cuando tenga oportunidad (también llamada *probabilidad de renuncia*). Como ejemplo, una ejecución donde tanto la probabilidad de activación como la de renuncia sean altas caracterizará un sistema donde los nodos generan muchos hijos para resolver tareas que surgen de su ejecución pero, que su tarea particular es, en sí, muy simple.

Estas dos probabilidades no son directamente conocidas por el *Worker*, se encuentran enmascaradas por una clase llamada *ActionGenerator* que genera acciones aleatorias de acuerdo a las probabilidades dadas y con distribución uniforme. Las acciones posibles serán la de activar

[1] por *Local Termination Manager*
[2] por *Global Termination Manager*

un nuevo hijo, terminar o continuar la ejecución.

El *GTM* generará los diferentes *ActionGenerators* con los parámetros dados por el programa principal y los enviará uno a uno en el momento de la configuración del sistema (justo antes de comenzar a resolver el problema). Cada nodo poseerá, entonces, su propio *ActionGenerator* y solicitará a éste las acciones que deba llevar a cabo.

La principal idea detrás de la inclusión de esta clase generadora de acciones se centra en la implementación de dos mecanismos diferentes de elección de acciones a seguir. El primero está basado directamente en la obtención de números al azar en tiempo de ejecución con las probabilidades ya mencionadas, el segundo, en cambio, se basa en la generación *a priori* de las acciones para que diferentes ejecuciones del sistema lo hagan con las mismas características de ejecución. Las acciones pre-generadas se almacenan en archivos ubicados en el nodo *Manager* que son leídos cuando el *GTM* crea uno a uno los generadores de acciones. De esta manera se logra que los *Workers* trabajen de manera transparente en ambos modos de generación de acciones.

En las siguientes dos secciones veremos cómo, cada uno de los modelos implementados, cumple esta especificación.

3. Modelo centralizado

Por su simpleza, el primero de los modelos que describiremos será el centralizado.

La configuración del sistema se compone de diferentes nodos trabajadores que llevarán a cabo las tareas para resolver el problema y un nodo administrador que teniendo conocimiento de todos los anteriores coordinará centralizadamente sus activaciones y desactivaciones.

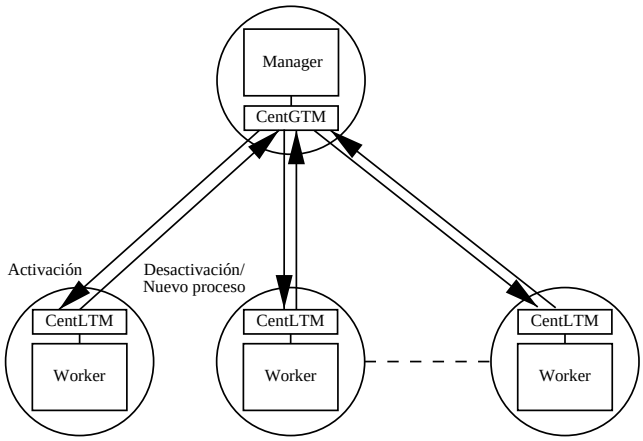

Figura 1: Terminación centralizada

Como puede verse en la Figura 1, cada nodo posee principalmente dos objetos que concurrentemente se dedican a llevar a cabo las tareas locales referidas a la solución del problema (*Worker*) y a la coordinación con el *Manager* (*CentLTM*, instancia del abstracto *LTM* que implementa su funcionamiento en este modelo).

Cuando un *Worker* decida activar otro proceso lo comunicará al *CentLTM* que reenviará la solicitud al *CentGTM* ubicado en el nodo administrador. El *CentGTM* poseerá de acuerdo a la especificación la información de la existencia de todos los nodos presentes en el sistema y su respectivo estado. Seleccionará según algún criterio propio cuál deberá activarse y enviará esta

señal al nodo elegido. Posteriormente, el *CentLTM* que recibe esta señal informa a su *Worker* que puede activarse. Por el contrario, si el *CentGTM* descubre que el sistema se encuentra en el tope de su capacidad y no hay nodos inactivos, descartará la solicitud.

Cuando un *Worker* decida desactivarse, lo informará a su *CentLTM* y este lo informará al *CentGTM*. Si este nodo es el último de los que se encontraban activos, el *CentGTM* lo notará chequeando su lista de nodos activos y luego finalizará enviando a todos los nodos un mensaje de finalización global para terminar su ejecución por completo.

4. Modelo distribuído

Los principales exponentes de modelos de terminación ditribuída se basan en estructuras de anillo.

En este trabajo llevamos a cabo una implementación de terminación por anillo basada en el modelo de dos pasadas [3]. Las variaciones se encuentran principalmente en la inclusión de un modelo de selección de nodos para su activación y un nuevo *token* (color rojo) destinado a informar a cada uno de los nodos que la terminación global es inminente y que deben finalizar su ejecución.

Los nodos se encuentran dispuestos en un anillo lógico, cada uno de ellos conoce al nodo siguiente y el último conoce al primero.

Al momento de la implementación y análogamente al modelo centralizado se crearon las clases *RingGTM* y *RingLTM* para el manejo de la detección de terminación y activación de los nodos que se encuentran presentes en el *Manager* y en los *Workers* respectivamente.

Existe un nodo distinguido que además de ser el responsable de cambiar el nodo de negro a blanco en cada vuelta, es el que recibe en primera instancia el nodo de terminación y el de activación. Para ser claros en este concepto veamos en detalle su funcionamiento:

Exiten dos tipos de *tokens* que difieren en la información que transportan y en la utilización que les da el sistema.

Token de activación Como adelantamos en la Sección 2 el modelo de activación de nodos en el sistema que implementa la detección de terminación distribuída será también distribuído y por consiguiente deberá funcionar eficientemente por sobre la estructura de anillo. Es por eso que este nuevo *token*, llamado de activación, se crea para permitir que una solicitud de activación viaje a través de los nodos buscando uno que, estando en estado de espera, pueda activarse. En el caso en que un token de este tipo recorra todo el anillo y alcance nuevamente el nodo que lo creó, esta solicitud de activación será eliminada y se dirá que el sistema sufrió un *descarte*. La información que este token contiene es simplemente el identificador de su creador.

Token de terminación Este tipo de token se utiliza para la detección de terminación. A diferencia del anterior, en donde un token por cada solicitud de activación viajará a traves del anillo, sólo existirá un token de terminación en todo el sistema. El funcionamiento es exactamente igual al de dos pasadas (ver [4]) con el agregado de un nuevo color. La información contenida dentro de este token es el color que puede ser uno de los siguientes:

Blanco: Según el algoritmo de terminación de dos pasadas un token será de color blanco cuando no haya pasado por ningún nodo que haya activado otro anterior en el anillo.

Negro: El token será de color negro cuando haya pasado por un nodo que ha activado otro anterior en el anillo.

Rojo: El token será de color rojo cuando todos los nodos estén en estado *inactivo*. Este token recorre una única vez el anillo. Lo hace justo antes de la finalización global del sistema para que los nodos terminen su ejecución por completo. Cabe destacar que el único nodo autorizado a cambiar el token de terminación a rojo es aquel distinguido en el principio del anillo.

El esquema de este modelo puede verse en la Figura 2.

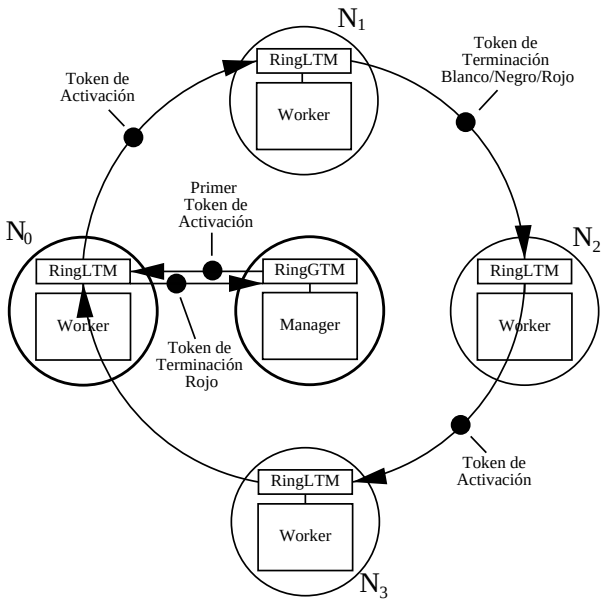

Figura 2: Terminación distribuída

5. Análisis comparativo de desempeño

5.1. Medidas y métricas

Las métricas de desempeño se calcularon en base a tres medidas tomadas sobre el desempeño de cada corrida. Son las siguientes:

Cantidad de activaciones (CA) La cantidad total de procesos que fueron activados exitosamente.

Cantidad de descartes (CD) Por haber implementado además del modelo de detección de terminación, un modelo de activación, decidimos tomar medidas con las que pudiéramos evaluarlo. La cantidad de descartes mide la cantidad de solicitudes de activación que no llegaron a activar ningún nodo.

Tiempo total de corrida (TT) Este tiempo se toma desde que el sistema recibe la primera solicitud de activación (aquella enviada por el *RingGTM* al nodo distinguido) hasta que el *token* rojo termina el recorrido por el anillo. Esta medida será expresada en segundos en todos los casos.

De acuerdo a estas tres medidas se calcularon las siguientes métricas:

Cantidad de pedidos (CP) Es la cantidad de solicitudes de activación por parte de los *Workers* que el sistema debió administrar, ya sea activando un nodo o descartándola. Se calcula según $CP = CA + CD$.

Pedidos por seg.($P_{\times S}$) Representa la cantidad de pedidos que el sistema administró por cada segundo de corrida. Para sistemas con la misma probabilidad de activación representará la eficiencia con la que el sistema administró los pedidos de los *Workers*.

Se calcula según:

$$P_{\times S} = \frac{CP}{TT}$$

Activaciones por seg ($A_{\times S}$) Representa la cantidad de activaciones que el sistema llevó a cabo por cada segundo de ejecución. Para sistemas con la misma probabilidad de activación representará la eficacia con que el sistema buscó nodos que pudieran cumplir la solicitud.

Se calcula según:

$$A_{\times S} = \frac{CA}{TT}$$

Porcentaje de descarte (PD) El porcentaje de pedidos que tuvieron que ser decartados. Debido a que lo que se busca es que el modelo de activación cumplimente la mayor cantidad de solicitudes posible esta medida representará el porcentaje en que el sistema logró este objetivo (la situación ideal es, por supuesto, que no existan descartes).

En todos los casos el acrónimo *Prom.* indicará que los valores expresados se obtuvieron como promedio de los resultados de las corridas realizadas. Asimismo *Var.* será la varianza de los valores.

5.2. Condiciones de corrida

Todas las corridas se llevaron a cabo sobre computadoras Pentium 200Mhz unidas por una red Ethernet de 10 Mbits. Para cada caso de estudio se llevaron a cabo 10 ejecuciones con los mismos parámetros de entrada y fueron descartadas aquellas instancias donde la cantidad de activaciones no superara la cantidad de nodos (salvo en el último caso de estudio donde se buscó exactamente lo contrario).

5.3. Caso de estudio 1

La simulación, en este caso, busca representar un sistema con una significativa cantidad de cálculo pero manteniendo solamente un proceso trabajador por nodo. Esto condiciona a que todas las comunicaciones entre procesos trabajadores se lleve a cabo a través de la red física de interconexión. Como puede verse en la Figura 3 la PR =0.05 y la PA =0.10 con 5 nodos sobre los cuales corre un único proceso trabajador respectivamente. La relación entre la probabilidad de activación y de renuncia es de 2 : 1 con lo cual el sistema tiende a tener varios procesos activos simultáneamente.

Los resultados obtenidos son los siguientes:

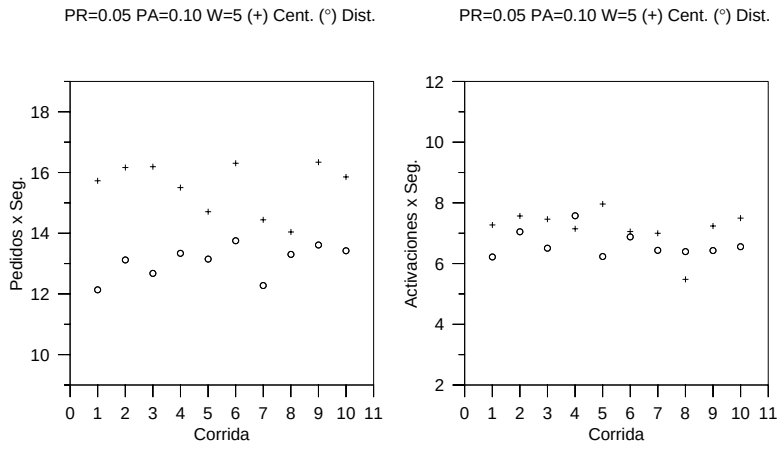

Figura 3: Pedidos y activaciones por segundo en el caso de estudio 1

Modelo	$Prom.P_{\times S}$	$Var.P_{\times S}$	$Prom.A_{\times S}$	$Var.A_{\times S}$	PD
Centralizado	15.52	0.63	7.16	0.38	53.83
Distribuído	12.88	0.90	6.40	0.68	50.45

El modelo centralizado supera claramente al distribuído en términos de eficiencia generales y en estabilidad pero sufre una leve caída en el porcentaje de activaciones que logra.

5.4. Caso de estudio 2

Este caso mantiene los parámetros probabilísticos del anterior, pero aquí la cantidad de procesos es duplicada manteniendo la cantidad de nodos, ver Figura 4. Este hecho incrementa la utilización de CPU y fuerza a que ciertas comunicaciones se lleven a cabo en los nodos de manera interna (cuando dos procesos en el mismo nodo necesitan comunicarse). Cabe destacar que esta condición se dará con mucha mayor frecuencia en el modelo distribuído. Debemos notar también, que cuando se intercambian mensajes dentro de un nodo, se puede dar un cierto grado de paralelismo de comunicación debido a que no colisionan con aquellos que se intercambian entre procesos internos de otro nodo, por lo que el modelo distribuído será beneficiado mientras que el centralizado se mantendrá invariante.

Los resultados obtenidos son los siguientes:

Modelo	$Prom.P_{\times S}$	$Var.P_{\times S}$	$Prom.A_{\times S}$	$Var.A_{\times S}$	PD
Centralizado	14.69	1.89	7.61	0.01	47.82
Distribuído	10.44	0.02	6.31	0.04	39.53

El beneficio obtenido por el paralelismo en el intercambio de mensajes no alcanza a contrarrestar el costo que impone el alto grado de utilización de CPU y la gran necesidad de comunicación dada por esta alta probabilidad de activación. Esto perjudica considerablemente al modelo distribuído, ya que son muy comunes los mensajes entre procesos adyacentes que en este caso se encuentran en el mismo nodo. A pesar de esto el sistema distribuído mantiene una varianza y un porcentaje de descarte menor.

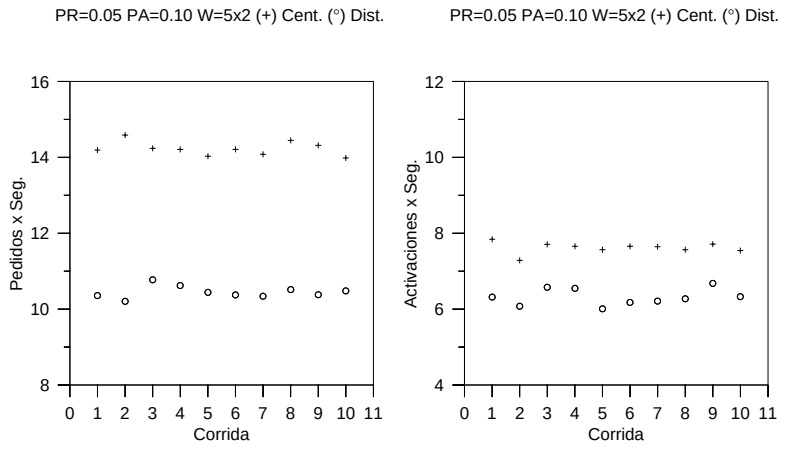

Figura 4: Pedidos y activaciones por segundo en el caso de estudio 2

5.5. Caso de estudio 3

Los parámetros probabilísticos en este caso buscan representar un sistema con alto grado de utilización de CPU y de baja generación de nuevas tareas, ver Figura 5. La comunicación será mínima y tendrá lugar en la capa física de red.

Los resultados obtenidos son los siguientes:

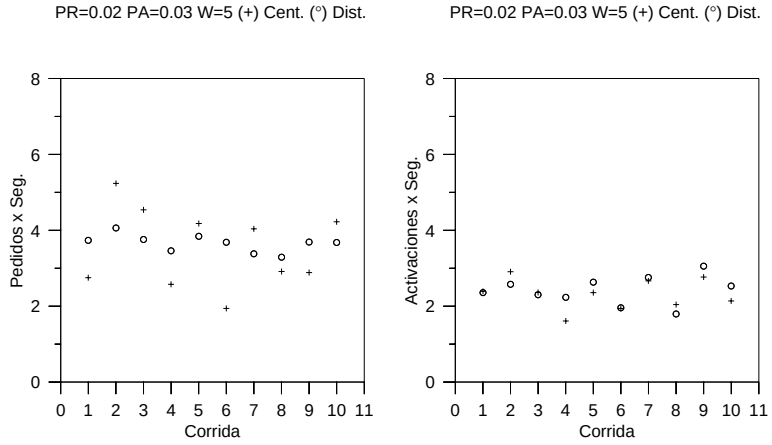

Figura 5: Pedidos y activaciones por segundo en el caso de estudio 3

Modelo	$Prom.P_{\times S}$	$Var.P_{\times S}$	$Prom.A_{\times S}$	$Var.A_{\times S}$	PD
Centralizado	3.31	1.18	2.17	0.22	30.43
Distribuído	3.65	0.04	2.41	0.12	33.81

Apreciamos que en este caso, el modelo distribuído supera levemente el desempeño del centralizado en términos de la cantidad de pedidos y activaciones por segundo, manteniendo su menor varianza pero aumentando el porcentaje de solicitudes descartadas.

5.6. Caso de estudio 4

De manera similar al caso anterior la diferencia entre la probabilidad de activación y la de renuncia es aquí, bastante leve, con una relación 3 : 2, como se observa en la Figura 6. Existirán en cada nodo dos procesos trabajadores que permitirán que exista paralelismo en la comunicación.

Los resultados obtenidos son los siguientes:

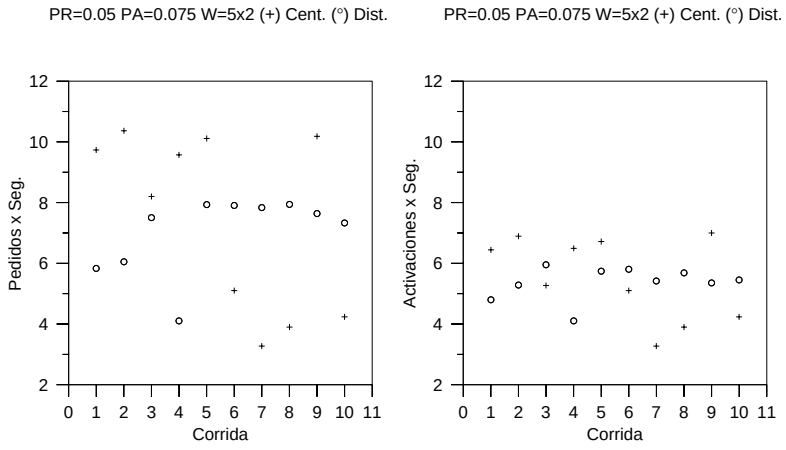

Figura 6: Pedidos y activaciones por segundo en el caso de estudio 4

Modelo	$Prom.P_{\times S}$	$Var.P_{\times S}$	$Prom.A_{\times S}$	$Var.A_{\times S}$	PD
Centralizado	7.44	7.84	5.51	1.65	20.06
Distribuído	7.00	1.47	5.35	0.27	22.00

A diferencia del *Caso 2* y por la menor probabilidad de activación que produce una menor demanda de comunicación, en este caso, el modelo distribuído se beneficia con el paralelismo en la comunicación y logra un mayor nivel de competitividad. Como en los casos anteriores la varianza se mantiene por debajo del centralizado pero el porcentaje de descarte lo supera.

5.7. Caso de estudio 5

Manteniendo las probabilidades anteriores, en este caso se busca obtener otra relación entre la utilización de CPU y la posibilidad de paralelismo de comunicación, ver Figura 7.

Los resultados obtenidos son los siguientes:

Modelo	$Prom.P_{\times S}$	$Var.P_{\times S}$	$Prom.A_{\times S}$	$Var.A_{\times S}$	PD
Centralizado	3.39	1.08	2.42	0.23	24.12
Distribuído	2.51	0.67	2.07	0.29	14.39

De la misma manera que en el *Caso 2* el costo de los mensajes internos es superior al beneficio que puede obtenerse por el paralelismo de comunicación en los nodos. Es por esto que vemos que el modelo distribuído cae considerablemente. Por otro lado el porcentaje de mensajes descartados es mayor en modelo centralizado y la varianza menor.

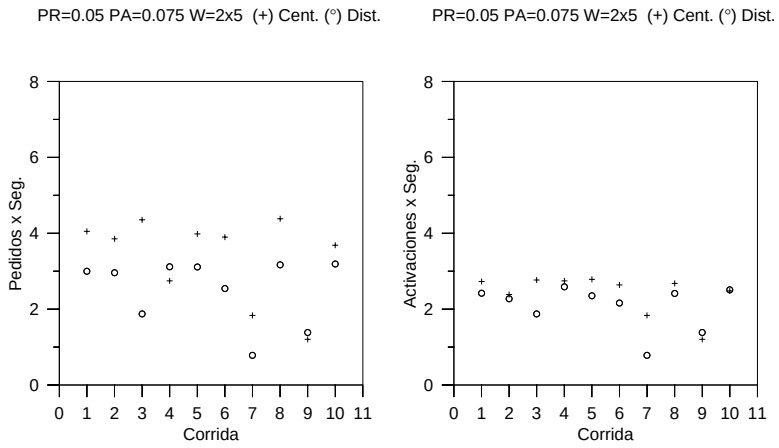

Figura 7: Pedidos y activaciones por segundo en el caso de estudio 5

5.8. Caso de estudio 6

Por último se buscó comparar los modelos en el caso extremo de máxima probabilidad de activación, es decir $PA = 1$. La probailidad de renuncia se fijo en $PR = 0.5$ para permitir que el sistema generara una buena cantidad de activaciones y alcanzara la condición de terminación en un tiempo razonable, ver Figura 8. Cada nodo tendrá un único proceso trabajador por lo que todas las comunicaciones se llevan a cabo en la capa física de red.

Los resultados obtenidos son los siguientes:

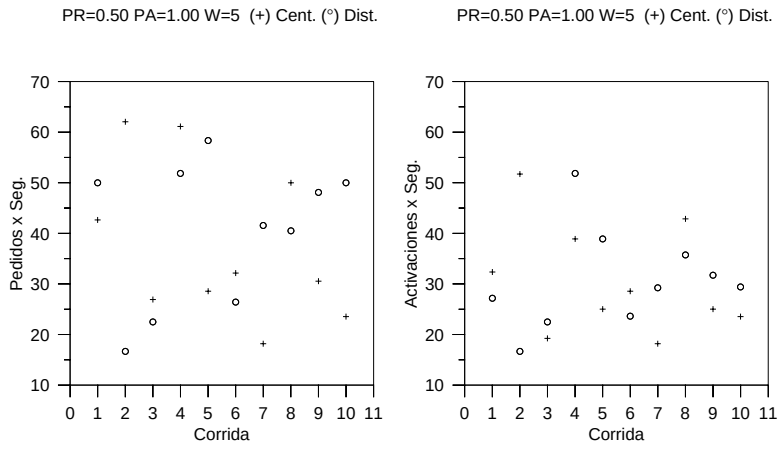

Figura 8: Pedidos y activaciones por segundo en el caso de estudio 6

Modelo	$Prom.P_{\times S}$	$Var.P_{\times S}$	$Prom.A_{\times S}$	$Var.A_{\times S}$	PD
Centralizado	37.46	213.01	30.44	103.20	16.18
Distribuído	40.74	183.86	30.78	89.06	20.60

Aquí apreciamos claramente el beneficio dado por el esquema de activación distribuída. El modelo centralizado requiere que todas las activaciones y desactivaciones sean administradas por un único proceso que en este caso, es saturado por la alta probabilidad de estos dos eventos.

6. Conclusiones

En este trabajo se presentaron e implementaron dos alternativas para activación y detección de terminación de procesos en sistemas distribuidos aplicando algoritmos centralizados y distribuidos. Luego de haber examinado diferentes casos en términos de las características de los ambientes en que ambas implementaciones fueron ejecutadas, logramos identificar aquellos aspectos que hacían más conducente la comparación.

Si bien debemos tener presente que en la totalidad de los casos de estudio presentados, el modelo distribuído no logra superar considerablemente al centralizado, debe notarse que todas las pruebas fueron realizadas sobre una red física de arquitectura Ethernet. Esto perjudica notablemente al modelo distribuído ya que todas las comunicaciones deben realizarse por medio de un único canal que debe ser compartido.

Lo destacable es que frente a una pequeña cuota de paralelismo en el tránsito de los mensajes entre procesos el modelo distribuído llega a recuperarse notablemente permitiendo aseverar que la configuración de único canal de comunicaciones es la principal causa de su menor performance.

Por otro lado, el sistema distribuído muestra una menor varianza en sus resultados, lo que lo dota de gran estabilidad. En lo concerniente al porcentaje de descartes descubrimos que en todos los casos aquel modelo que logra un mejor desempeño sufre una mayor proporción de descarte de solicitudes, lo que es comprensible frente a un sistema que se encuentra al tope de su capacidad la mayor parte del tiempo.

Podemos concluir entonces que para sistemas donde la cantidad de nodos sea pequeña y la arquitectura física de comunicación se base en un único canal que debe ser compartido sin importar la procedencia y destino de los mensajes, el sistema centralizado es la mejor alternativa. Por otro lado cuando la cantidad de procesos crece y la necesidad de comunicación se hace mayor, el modelo distribuído se transforma en una alternativa viable, lo que nos permite suponer que sobre una arquitectura de comunicación basada en anillo y una configuración de los nodos acorde, este último modelo será el único a tener en cuenta.

Referencias

[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.