SISTEMAS DE CÓMPUTO DE ALTAS PRESTACIONES CON ALTA DISPONIBILIDAD: EVALUACIÓN DE LA PERFORMANCE EN DIFERENTES CONFIGURACIONES

Cecilia M. Lasserre[1], Emilio Luque Fadón[2], Dolores I. Rexachs del Rosario[2], Nilda M.Pérez Otero[1], Sandra Adriana Mendez[1], C. Marcelo Pérez Ibarra[1], Héctor P. Liberatori[1], Luis M. Valdiviezo[1], Fernando E. Flores[1], Rodrigo Cachambe[1]

[1]Facultad de Ingeniería – Universidad Nacional de Jujuy – Argentina
{lasserre, nilperez, smendez, cmperezi, hliberatori, mlvaldiviezo, efloresc, rcachambe}@fi.unju.edu.ar
[2]Departamento de Arquitectura de Computadores y Sistemas Operativos (DACSO-CAOS) – Universidad Autónoma de Barcelona – España
{dolores.rexachs, emilio.luque}@uab.es

CONTEXTO

El presente proyecto pertenece al GIS (Grupo de Ingeniería del Software) de la Facultad de Ingeniería de la UNJu y pertenece al Programa 08/D058. Se encuadra en el campo del Cómputo de Altas Prestaciones, específicamente en Tolerancia a Fallos.

RESUMEN

La Computación de Altas Prestaciones a través de clusters de computadores basados en Workstation y redes convencionales, posibilitó la construcción y uso de computadores paralelos. En las arquitecturas paralelas el objetivo principal es el aumento de prestaciones, utilizando el potencial ofrecido por gran número de procesadores.

La construcción de un cluster tiene tres retos: Alto Rendimiento, Alta Disponibilidad y Alta Productividad. Manejar eficientemente un número elevado de computadores en ambientes heterogéneos no es trivial, por ello requiere un cuidadoso diseño de la arquitectura y funcionalidad. Respecto a la alta disponibilidad, debemos considerar la probabilidad de los fallos o desconexión de nodos, reduciendo el tiempo medio entre fallos del computador paralelo como un todo.

Este proyecto pretende definir la configuración adecuada de tolerancia a fallos para diferentes tipos de aplicaciones, teniendo en cuenta los requerimientos de rendimiento y prestaciones del usuario, definir y validar un modelo genérico de aplicación-prestación-tolerancia a fallos.

Palabras clave: *Computación de Altas Prestaciones, HPC, Alto Rendimiento, Alta Disponibilidad, Alta Productividad, Tolerancia a Fallos*

1. INTRODUCCION

1.1 Clusters

El número de computadores que se interconectan en red en la actualidad está creciendo, por otro lado, el software que se desarrolla cada vez exige más y más recursos de los computadores, como procesadores, memoria, almacenamiento,… [1]. La llegada de la Computación de Altas Prestaciones (*HPC - High Performance Computing*) a través de *clusters* de computadores basados en *Workstation* (WS) y redes convencionales (local o remota) o COTS (*Commodity Off The Shelf*), ha posibilitado que muchas instituciones y organizaciones vean viable la construcción y el uso de computadores paralelos. En las llamadas arquitecturas paralelas el objetivo principal es el aumento de la capacidad de procesamiento, utilizando el potencial ofrecido por un gran número de procesadores (alto aumento del *speedup* y alta disponibilidad) [2].

La construcción de un cluster tiene tres retos: Alto Rendimiento (*High Performance: HP*), Alta Disponibilidad (*High Availability: HA*) y Alta Productividad (*High Throughput: HT*) [1] [3]. Manejar eficientemente un número elevado de computadores en ambientes heterogéneos (heterogeneidad de nodos de cómputo, de redes, etc.) no es trivial, por ello requiere un cuidadoso diseño de la arquitectura y su funcionalidad. Respecto a la alta disponibilidad, cuando se considera un elevado número de nodos funcionando durante un tiempo elevado, debemos considerar que los fallos en los nodos o desconexión son eventos probables, reduciendo el MTBF (*Mean Time Between Failures* - tiempo medio entre fallos) del computador paralelo como un todo [4].

Los clusters se construyen a partir de varios nodos procesadores independientes conectados por alguna tecnología de red. Esos sistemas se diferencian de las máquinas masivamente paralelas por el débil acoplamiento entre sus nodos, o sea, los elementos del cluster no tienen acceso a una memoria común compartida. Otra característica que marca los clusters es la inexistencia de un reloj común que pueda ser utilizado para ordenar los eventos. Toda la comunicación entre los nodos se da a través de paso de mensajes a través de canales de comunicación. Además, son generalmente construidos con elementos heterogéneos y asíncronos. La tolerancia a fallos tiene el objetivo de proveer al cluster de una capacidad de operación continuada, intentando que sólo provoque una caída pequeña de prestaciones ocasionada por la prevención y la detección de fallos, tanto en la presencia como en la ausencia de

fallos. A pesar de conocerse un buen conjunto de técnicas de tolerancia a fallos, su aplicación en los clusters es muy compleja y sus resultados son muchas veces insatisfactorios debido al *overhead* introducido en el sistema [5].

1.2 Tolerancia a Fallos

Los clusters presentan una redundancia física intrínseca debido a la existencia de numerosos nodos de cómputo, que puede ser utilizada para el empleo de tolerancia a fallos. La ocurrencia de fallo en algún nodo procesador no tiene que significar necesariamente la interrupción del suministro del servicio. Una solución posible para esto es que el sistema sea reconfigurado, utilizando solamente los nodos disponibles.

Con el objetivo de aumentar la garantía de funcionamiento de los clusters, son esenciales, técnicas de prevención de fallos, detección de errores, diagnóstico, recuperación y reconfiguración.

Es necesario definir una arquitectura que permita ejecutar aplicaciones paralelas que requieren HP, HA y HT. Esta arquitectura debe garantizar que se realice la ejecución eficiente de una aplicación paralela cuando se utiliza un entorno con un elevado número de nodos, durante un largo período de tiempo. Por otro lado, es necesario considerar que la probabilidad de fallo aumenta, y puede llegar a ocurrir que el fallo ocasione la pérdida total del trabajo realizado. Un cluster de alta disponibilidad intenta mantener en todo momento la prestación de servicio encubriendo los fallos que se pueden producir.

Por lo tanto para proveer HA es necesario utilizar técnicas de tolerancia a fallos [6] que permitan garantizar (confiabilidad o garantía de servicio) Fiabilidad (*Reliability*), Disponibilidad (*Availability*) y Facilidad de mantenimiento (*Serviceability*) del sistema.

La disponibilidad de un sistema se puede ver afectada por diferentes averías de sus componentes, a pesar de utilizar componentes cotidianos sin características especiales de fiabilidad nos interesa disponer de un sistema fiable.

Si ocurre un error durante la ejecución y ocasiona una ejecución incorrecta de las funciones del sistema, se tiene una avería [7] [8]. Estos pasos, no se producen de forma simultánea en el tiempo, sino que existe un tiempo de inactividad, llamado "latencia del error" desde el instante en que se produce el fallo hasta que se manifiesta el error. [9]

En caso de aparición de fallo en algún componente, la tolerancia a fallos busca que el sistema siga trabajando, aunque esté funcionando en modo degradado, hasta que acabe la ejecución, lo que **podrá incidir en mayor o menor medida en sus prestaciones**. El número de fallos que pueden estar presentes en un momento dado dependerá del número de componentes del sistema, del *MTBF* y del tiempo de ejecución de la aplicación. La probabilidad de que dos o más fallos ocurran simultáneamente decrece. La tolerancia a fallos en un sistema se logra mediante la inclusión de técnicas de redundancia [10]. Puede haber redundancia en cualquier nivel: utilización de componentes hardware extra (redundancia en el hardware), repetición de las operaciones y comparación de los resultados (redundancia temporal), codificación de los datos (redundancia en la información) e incluso la realización de varias versiones de un mismo programa (redundancia de software). Las técnicas más utilizadas en computación paralela son el uso de técnicas de replicación de datos (redundancia de información) [8] o de *checkpoint* (redundancia temporal) [11].

Para evitar la interrupción en el suministro del servicio, debido algún fallo en sus componentes, los fallos deben ser detectados lo más rápidamente posible: latencia del fallo.

El nodo en que ha ocurrido el fallo debe ser identificado a través de diagnóstico apropiado y finalmente reparado o aislado a través de reconfiguración del sistema. Esa reconfiguración se hace reasignando tareas y seleccionando caminos alternativos de comunicación entre los nodos.

Los fallos en clusters pueden ser causados por problemas en el hardware, sistema operativo (OS), aplicaciones, o por el propio software de tolerancia a fallos [12].

Como vemos es necesario utilizar alguna técnica o mecanismo que proteja al sistema, utilizamos redundancia temporal, protocolos de *rollback-recovery*, debemos considerar la prevención (generar redundancia), la detección (monitorización), el diagnóstico (latencia del error) y la recuperación y reconfiguración del sistema (retornar el sistema a condiciones operativas razonables después un fallo) como un todo. Para lograr esto la tolerancia a fallos incluye las etapas:

- Protección: redundancia
- Detección del error
- Estimación de daños: diagnosis
- Recuperación del error: llevar al sistema a un estado correctos, desde el que pueda seguir funcionando (tal vez con funcionalidad parcial)
- Tratamiento del fallo y continuación del servicio del sistema: consistencia global del sistema: reconfiguración y enmascaramiento del fallo.

Se debe tratar de mantener el equilibrio entre la latencia y el *overhead* que introducen las técnicas de tolerancia a fallos. Además de aumentar el coste del sistema, desde el punto de vista del usuario, las actividades realizadas por el mecanismo de

tolerancia a fallos reducen las prestaciones del sistema.

1.3 RADIC

Se parte de una arquitectura tolerante a fallos RADIC (***R**edundant **A**rray of **D**istributed **I**ndependent Fault Tolerance **C**ontrollers*) escalable, flexible, transparente y descentralizada para sistemas de paso de mensajes, cuyo objetivo principal es asegurar que una aplicación paralela-distribuida finalice correctamente aun si ocurrieron fallos en algunos nodos del computador paralelo.

RADIC establece un modelo de arquitectura que define la interacción de la arquitectura tolerante a fallos y la estructura del computador paralelo, implementa una capa entre el nivel de paso de mensajes y la estructura del computador.

El núcleo de la arquitectura RADIC es un controlador completamente distribuido para tolerancia a fallos que trata automáticamente los fallos en los nodos del cluster. Tal controlador comparte los recursos del computador paralelo usados en la ejecución de la aplicación paralela.

RADIC adopta el mecanismo de *rollback-recovery* basado en protocolo *log* pesimista, realiza automáticamente un grupo de actividades requeridas para la Tolerancia a Fallos. Cada actividad está dentro de alguno de los cuatro procedimientos generales requeridos por un mecanismo automático de tolerancia a fallos. La estructura de la arquitectura de RADIC usa un grupo de procesos que colaboran para crear un controlador distribuido para tolerancia a fallos. Hay dos clases de procesos: protectores y observadores. Cada nodo del computador paralelo tiene un protector dedicado y hay un observador dedicado asociado a cada proceso de la aplicación paralela. En la Tabla 1.1 se presentan las Fases Operativas de RADIC y el componente RADIC implicado en cada fase.

Tabla 1.1: Fases Operativas de RADIC

Fase Funcional	Actividad	Componente RADIC
Protección: Salvar el estado	• Realizar y enviar los *checkpoints* • Almacenar *checkpoints* • Realizar y enviar *logs* • Almacenar *logs* de mensajes • Gestionar basura (*checkpoints* obsoletos).	• Observador • Protector • Observador • Protector • Protector
Detección de fallos	• Monitorizar los nodos usando un mecanismo de *heartbeat/watchdog* • Detectar los fallos de comunicación entre procesos.	• Protector • Observador
Recuperación	• Reiniciar los procesos fallidos desde los *checkpoints* y *logs*. • Reejecutar y procesar los *logs*.	• Protector • Observador
Reconfiguración y enmascaramiento de fallos	• Asegurar que los procesos que no se vieron afectados por el fallo puedan encontrar los procesos recuperados.	• Observador y Protector

Estas fases suceden concurrentemente con la ejecución de la aplicación paralela y no interfiere en sus resultados [13].

2. LINEAS DE INVESTIGACION y DESARROLLO

- Definir la configuración adecuada de tolerancia a fallos para diferentes tipos de aplicaciones, teniendo en cuenta los requerimientos de rendimiento y prestaciones del usuario.
- Definir un modelo genérico de aplicación-prestación-tolerancia a fallos.
- Validar experimentalmente el modelo propuesto.
- Formar recursos humanos en HPC en general y Tolerancia a fallos, en particular.

3. RESULTADOS OBTENIDOS/ESPERADOS

Debido a que el presente proyecto está previsto para el trienio 2009-2011, actualmente se encuentra en la transición entre una etapa de recopilación, selección y análisis de la bibliografía existente sobre tolerancia a fallos, RADIC y simuladores de protocolos y tecnologías de red y la selección de una aplicación piloto, selección y configuración de un simulador de protocolos y tecnologías de red y configuración inicial y prueba de RADIC.

Para los próximos meses se espera iniciar la etapa de configuración de RADIC sobre el cluster real y simulado para la aplicación seleccionada y la experimentación sobre el simulador y cluster real, sin y con tolerancia a fallos con el fin de realizar una contrastación de las prestaciones y medidas de rendimiento obtenidas de la simulación y la ejecución en el cluster.

A futuro se realizarán las etapas de:

- Selección de un conjunto de aplicaciones con diferentes requisitos de rendimiento y prestaciones. Configuración del mecanismo de tolerancia a fallos RADIC para las aplicaciones seleccionadas.
- Caracterización del conjunto de aplicaciones seleccionadas y experimentación sobre el simulador, sin y con tolerancia a fallos (RADIC).
- Formulación del Modelo Aplicación-Prestación-Tolerancia-a-Fallos.
- Validación experimental del Modelo Aplicación-Prestación-Tolerancia-a-Fallos.

4. FORMACION DE RECURSOS HUMANOS

Durante el desarrollo del proyecto se tiene previsto realizar transferencia a los alumnos de la carrera de Ingeniería en Informática de la UNJu y a los alumnos de Maestría en Ingeniería del Software y de Doctorado en Informática de la Red UNJu-UNSL.

Además, tres de los integrantes del proyecto están realizando el trabajo final de la Especialización en Cómputo de Altas Prestaciones y Tecnología GRID y posteriormente trabajarán en su tesis Doctoral sobre esta temática. También se ofrecerán opciones de trabajos finales de carrera en HPC a los alumnos de Ingeniería en Informática.

5. BIBLIOGRAFIA

[1] Buyya R. High Performance Cluster Computing: Architectures and Systems, Volume 1 [Book]. - [s.l.] : Prentice Hall PTR, 1999.

[2] Rodrigues de Souza Josemar FTDR: Tolerancia a fallos, en clusters de computadores geográficamente distribuidos, basada en Replicación de Datos [Book]. - Barcelona : [s.n.], 2006.

[3] Sterling T.L. Launching into the Future of Commodity Cluster [Conference] // IEEE International Conference on Cluster Computing (CLUSTER 2002). - 2002. - p. 345.

[4] Bosilca G. [et al.] MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes [Conference] // SC '02: Proceedings of the Proceedings of the IEEE/ACM SC2002 Conference. - 2002.

[5] Rodrigues de Souza J [et al.] Fault Tolerant Master-Worker over a Multi-Cluster Architecture [Conference] // ParCo2005. - 2005.

[6] Avizienis A. Toward systematic design of fault-tolerant systems [Journal] // IEEE, Computer v.30, Iss.4. - 1997. - pp. 51-58.

[7] Pradhan D.K. Fault Tolerant System Design [Book]. - New Jersey : Prentice Hall, 1996.

[8] Weissman J.B. Fault Tolerant Computing on the Grid: What are My Options? [Conference] // High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on. - 1999.

[9] Johnson B.W. Design and Analysis of Fault Tolerant Digital Systems [Book]. - [s.l.] : Addison Wesley, 1990.

[10] Avizienis A., Laprie J. and Randel B. & Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing [Journal] // IEEE Transactions on Dependable and Secure Computing v 1, n.1. - 2004. - pp. 11-33.

[11] Chakravorty S. & Kalé, L.V. A Fault Tolerant Protocol for Massively [Conference] // 18th International Parallel and Distributed Processing Symposium (IPDPS'04) - Workshop 11. Publisher: IEEE Computer Society. - 2004.

[12] Sun H. and Han J.J. & Levendel, H. A Generic Availability Model for [Conference] // PRDC '01: Proceedings of the 2001 Pacific Rim International Symposium on Dependable Computing. - Washington : IEEE Computer Society, 2001.

[13] Duarte Angelo Amancio RADIC: A Powerful Fault-Tolerant Architecture [Book]. - Barcelona : [s.n.], 2007.