Desarrollo de un Entorno de Simulación basado en OMNeT++ para RADIC

Nilda M. Pérez Otero[1], Jussara de Marques Almeida[2], Dolores I. Rexachs[3], Germán Montejano[4]

[1]Facultad de Ingeniería – Universidad Nacional de Jujuy (UNJu)

[2]Departamento de Ciência da Computacão – Universidade Federal de Minas Gerais (UFMG)

[3]Departamento de Arquitectura de Computadores y Sistemas Operativos – Universidad Autónoma de Barcelona (UAB)

[4]Departamento de Informática – Universidad Nacional de San Luis (UNSL)

nilperez@fi.unju.edu.ar, jussara@dcc.ufmg.br, dolores.rexachs@uab.es, gmonte@unsl.edu.ar

Resumen. *Los sistemas de Cómputo de Altas Prestaciones se utilizan para desarrollar software en una gran cantidad de campos. Actualmente es evidente el creciente predominio e impacto de las aplicaciones de HPC en la sociedad moderna. Sin embargo, la presencia de fallos en el hardware o software de computadores paralelos genera nuevas necesidades en el uso de mecanismos tolerantes a fallos para asegurar que las aplicaciones finalicen exitosamente. Es por ello que se ha desarrollado RADIC, una arquitectura transparente, descentralizada, flexible y escalable para tolerancia a fallos en sistemas de paso de mensajes que provee alta disponibilidad. Uno de las dificultades que sufren los desarrolladores de RADIC es realizar pruebas en grandes clusters y sin verse limitados por una implementación específica de MPI. Es por ello que la prensente línea de investigación propone el desarrollo de un entorno de simulación basado en OMNeT++ para RADIC.*

Contexto

El presente proyecto se desarrolla como tesis de Maestría en Ingeniería de Software de la Facultad de Ciencias Físico, Matemáticas y Naturales de la Universidad Nacional de San Luis. Esta línea de investigación está inserta en el proyecto ***Simulador de un cluster tolerante a fallos basado en OMNeT++*** que a su vez forma parte del proyecto ***Sistemas de Cómputo de Altas Prestaciones con Alta Disponibilidad: Evaluación de la Performance en Diferentes Configuraciones*** que está acreditado y financiado por la Secretaria de Ciencia y Técnica y Estudios Regionales de la Universidad Nacional de Jujuy (SECTER-UNJu) e incluido en el programa de incentivos al docente-investigador bajo el código 08-D0093.

1. Introducción

Los sistemas de Cómputo de Altas Prestaciones (*High Performance Computing* - HPC) se utilizan para desarrollar software en una gran cantidad de campos, incluyendo física nuclear, simulación de accidentes, procesamiento de datos de satélites, dinámica de fluidos, modelado del clima, bioinformática y modelado financiero. El TOP500 [1] lista los 500 mejores sistemas de cómputo de altas prestaciones. La gran variedad de organizaciones científicas, gubernamentales y comerciales presentes en esta lista ilustra el creciente predominio e impacto de las aplicaciones de HPC en la sociedad moderna. [Car07].

La presencia de fallos en el hardware o software de computadores paralelos genera nuevas necesidades en el uso de mecanismos tolerantes a fallos para asegurar que las aplicaciones finalicen exitosamente. Hasta hace poco tiempo los computadores paralelos en producción no se veían afectados por fallos con la suficiente frecuencia como para justificar el uso sistemático de mecanismos de tolerancia a fallos. Sin embargo, la situación está cambiando y los fallos se hacen cada vez más frecuentes con cada nueva generación de computadores masivamente paralelos, *clusters* cada vez más grandes y complejos con necesidades de tiempos de cómputo ininterrumpido más largos o requerimientos de disponibilidad de 24 horas los 7 días de la semana. Hay una clara tendencia hacia una situación en la que los fallos no serán eventos aislados sino tenderán a ser eventos ordinarios en dichos sistemas.

[1]http://www.top500.org

Recientemente, algunos centros de supercomputadores han publicado estadísticas acerca de fallos [Cap09]. La mayoría de estas estadísticas están basadas en datos de confiabilidad, disponibilidad y servicio (*reliability*, *availability*, *serviceability* - RAS) provistas por Los Alamos National Laboratory (LANL), National Energy Research Scientific Computing Center (NERSC), Pacific Northwest National Laboratory, Sandia National Laboratory (SNL) y Laurence Livermore National Laboratory (LLNL) en los Estados Unidos de América. LANL proveyó información muy detallada con una descripción de 23.000 eventos que causaron paradas en la aplicación. Los datos de LANL corresponden a 22 *clusters* con hasta 4.096 CPUs, durante un período de 10 años. Esto representa alrededor de 5.000 nodos de cómputo y un total de 24.000 de CPUs (algunos nodos son multiprocesador). Schroeder y Gibson [SG07] presentan un análisis estadístico del conjunto de datos del LANL RAS donde se puede observar que el número de fallos por año puede ser tan bajo como 100 pero también puede exceder los 1000 para algunos sistemas. Para estos últimos, tres fallos por día implican que las aplicaciones largas que utilizan todos los nodos de cómputo y demoran más de 8 horas tienen pocas posibilidades de finalizar correctamente. El análisis estadístico también demuestra que el número de fallos por procesador en los diferentes sistemas se mantuvo bastante estable desde 1996 hasta 2004 lo que sugiere que el tiempo medio de interrupción (Mean Time To Interrupt - MTTI) del procesador tiene pocas posibilidades de mejorar significativamente en un futuro cercano.

Las estadísticas anteriores llevan plantear la necesidad de implementar un sistema tolerante a fallos para HPC. En orden a alcanzar alta disponibilidad, tal sistema de tolerancia a fallos debe proveer una recuperación y detección de fallos automática y transparente. Aun en estos casos, las interrupciones de servicio (por ejemplo, una parada completa del programa en ejecución) pueden ocurrir si no existen nodos de reemplazo o la degradación del sistema generada por los fallos llega a niveles inaceptables. Con el fin de asegurar un MTTI mayor, tales soluciones también deben proveer los medios para restaurar la configuración original del sistema (número inicial de nodos de reemplazo, o la distribución de procesos por nodo), sin detener la ejecución de la aplicación. Además, para una tolerancia a fallos reactiva es también muy deseable que pueda realizar tareas de mantenimiento preventivo, como, por ejemplo, reemplazar máquinas susceptibles a fallos sin interrupciones al sistema. [SDRL08]

1.1. Arquitectura RADIC

Considerando estos aspectos, Duarte *et al.* han propuesto y desarrollado RADIC (***R**edundant **A**rray of **D**istributed **I**ndependent Fault Tolerance **C**ontrollers*) [DRL06]. RADIC [DRL07] es una arquitectura transparente, descentralizada, flexible y escalable para tolerancia a fallos en sistemas de paso de mensajes que provee alta disponibilidad.

RADIC basa su operación en el mecanismo de *rollback-recovery* basado en protocolo log pesimista. En tal protocolo, se realizan *checkpoints* regularmente y todos los mensajes recibidos por cada proceso de la aplicación deben ser registrados por el receptor.

Como sistema transparente de tolerancia a fallos, RADIC realiza automáticamente una serie de actividades requeridas por el protocolo de *rollback-recovery*. Es por ello que cada actividad de RADIC se encuentra enmarcada en uno de los cuatro procedimientos generales de un mecanismo transparente de tolerancia a fallos:

- Protección: salvar el estado
- Detección de Fallos
- Recuperación
- Reconfiguración y Enmascaramiento de Fallos

Para llevar a cabo estas tareas, RADIC tiene dos entidades que trabajan juntas: ***observadores*** y ***protectores***. Cada nodo del computador paralelo tiene un protector dedicado y hay un observador dedicado asociado a cada proceso de la aplicación paralela. Estas entidades realizan log pesimista de mensajes basado en receptor y *checkpointing* no coordinado de manera de recuperar la aplicación paralela de un fallo, y un mecanismo de *heartbeat/watchdog* para detectar fallos. A continuación se describen

en más detalle los componentes funcionales de RADIC.

RADIC presenta dos niveles de protección: un nivel básico, donde provee sus funcionalidades sin la necesidad de recursos pasivos usando algunos de los nodos activos de la actual configuración para recuperar cualquier proceso fallido que pudiera degradar el rendimiento, adecuado sólo para aplicaciones de corto tiempo de ejecución, o bien, aplicaciones que puedan tolerar pérdida de recursos, como aquellas con balanceo de carga dinámico y un nivel de protección destinado a aplicaciones que demandan un comportamiento *non-stop*. En este nivel RADIC provee una redundancia dinámica flexible a través del manejo transparente de nodos *spare*.

Con el objetivo de validar las distintas implementaciones de RADIC se ha realizado mucha experimentación en distintos sistemas reales, con distinto hardware y distintas implementaciones de MPI. Si bien estos estudios permitieron obtener información acerca de varios de los parámetros de RADIC, estos experimentos en cluster reales están limitados tanto por el tamaño del cluster o por las implementaciones concretas de MPI que se utilizan. Por un lado, realizar pruebas en diferentes configuraciones y tamaños de *clusters* es un proceso arduo y costoso y por otro lado, desarrollar diferentes implementaciones de RADIC es costoso y consume mucho tiempo. Por este motivo, es fundamental encontrar métodos mas rápidos y baratos de ejecutar estas pruebas.

1.2. Simulación en HPC

Una solución a los problemas presentados al final de la sección anterior sería simular las configuraciones y la arquitectura RADIC para obtener aproximaciones sobre el comportamiento real de RADIC.

Actualmente, es cada vez más frecuente el uso de modelos de simulación computacional en HPC, ya sea como ayuda modelado de prestaciones, [DLWJ08] como para explorar arquitecturas o aplicaciones [HMS+09] [MR09] o como una herramienta de predicción de tráfico [TLCS09]

En la literatura se encuentra muchos trabajos centrados en simular grandes redes y aplicaciones de HPC. La mayoría de los simuladores de redes están centrados en arquitecturas específicas. En [MSSD00] se presenta otro simulador de SANs (Redes de Área de Almacenamiento) que permite trabajar tanto con trazas de tráfico reales como sintéticas, y simula fallos en enlaces y *switches*, canales virtuales, diferentes algoritmos de ruteo, etc. PARSEC [BMT+98] es un entorno de simulación de eventos discretos, que, mediante un compilador mejorado de C++ permite simular entidades y constructores de mensajes de comunicación entre entidades. SIMCAN [NFG+08] es un entorno de simulación para grandes redes complejas de almacenamiento que permite simular estas redes y sus subsistemas subyacentes correspondientes (I/O, *Networking*, etc.).

También es posible encontrar entornos de simulación de redes de propósito general que permiten crear diferentes configuraciones de redes, con diferentes tipos de nodos, *switches*, topologías, protocolos, etc. Ejemplos de éstos son OPNET Modeler [2] y OMNeT++ [3].

2. Líneas de Investigación y desarrollo

En la búsqueda de una alternativa menos costosa y más viable de analizar las características y parámetros de RADIC se plantea el objetivo principal de este proyecto que consiste en desarrollar un *framework* basado en OMNeT++ donde se puedan simular distintos esquemas de tolerancia a fallos y en particular que permita implementar los módulos de la arquitectura de RADIC. Este simulador permitirá un mejor análisis y comprensión de las funciones de RADIC interactuando con el sistema de cómputo y con las aplicaciones. Es decir, el simulador:

- permitirá el desarrollo y prueba de nuevas políticas en sistemas que no están disponibles físicamente,
- permitirá el análisis del comportamiento del sistema (desbalanceo de carga, cuellos de botellas causados por fallos) mediante la inyección de diferentes patrones de fallos,

[2] http://www.opnet.com/

[3] http://www.omnetpp.org/

- ayudará en el proceso de toma de decisiones (por ejemplo, si se detecta un cuello de botella: cuántos nodos spare serán necesarios, dónde ubicarlos, cuál es la influencia del mapeo entre protectores y observadores) permitiendo así la evaluación de diferentes configuraciones de RADIC.

En el ámbito de este trabajo el simulador se utilizará específicamente para simular y analizar una de las fases de RADIC: la fase de detección. Esta fase involucra el mecanismo de *heartbeat/watchdog* realizado los protectores. En este nivel la simulación abarcará la comunicación entre protectores en tiempo real, la gestión de las estructuras de datos de los módulos de RADIC y el mecanismo de *heartbeat/watchdog*. Modelar esta fase específica de la tolerancia a fallos permitirá comenzar con la validación del modelo completo de RADIC, porque se estarán utilizando sus estructuras de datos básicas, la comunicación entre procesos RADIC, operaciones en tiempo real, etc.

3. Resultados Obtenidos/Esperados

La tesis a la que corresponde esta línea de investigación tiene planificadas las siguientes tareas: Estudio del entorno de simulación OMNeT, Estudio Detallado de RADIC, Diseño del Entorno de Simulación, Implementación y Validación del Entorno de Simulación para RADIC, Análisis y Diseño de las Funcionalidades RADIC a ser simuladas (correspondientes a la fase de Detección) y Simulación y Análisis de Resultados. Actualmente se está diseñando el entorno de simulación para RADIC, mientras en paralelo se generan trazas en un cluster real que servirán para el diseño de las cargas reales para validar en entorno y sintéticas para realizar las simulaciones. Una vez que el entorno de simulación esté validado se podrán ejecutar las simulaciones que se espera permitan determinar qué parámetros podrían configurarse durante la experimentación de la siguiente fase, por ejemplo variar el parámetro que especifica el número de fallos de *heartbeats* consecutivos necesarios antes de iniciar el procedimiento de recuperación o el parámetro que especifica el tiempo de espera máximo de *watchdog* que inicia el procedimiento de recuperación. Este será el primer paso en el desarrollo del modelo de *overhead* introducido por la arquitectura RADIC (en este caso el *overhead* causado por el mecanismo de *heatbeat/watchdog*)

4. Formación de Recursos Humanos

Actualmente, dentro de esta línea de investigación se están realizando una Tesis de Maestría y una Tesina de Grado de Ingeniería en Informática de la UNJu. Además se prevee la continuación de esta misma línea de proyecto como tesis doctoral del tesista y una mayor interacción con la Universidade Federal de Minas Gerais y la Universidad Autónoma de Barcelona. Adicionalmente, se espera que otras tesis de Maestría, así como tesinas de grado surjan a partir de los logros obtenidos en la presente línea.

Referencias

[BMT+98] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu-an Chen, Xiang Zeng, Jay Martin, and Ha Yoon Song. Parsec: A parallel simulation environment for complex systems. *Computer*, 31(10):77–85, 1998.

[Cap09] Franck Cappello. Fault Tolerance in Petascale/ Exascale Systems: Current Knowledge, Challenges and Research Opportunities. *International Journal of High Performance Computing Applications*, 23(3):212–226, 2009.

[Car07] Jeffrey C. Carver. Third international workshop on software engineering for high performance computing (hpc) applications. In *ICSE COMPANION '07: Companion to the proceedings of the 29th International Conference on Software Engineering*, page 147, Washington, DC, USA, 2007. IEEE Computer Society.

[DLWJ08] Wolfgang E. Denzel, Jian Li, Peter Walker, and Yuho Jin. A framework for end-to-end simulation of high-performance computing systems. In *Simutools*

'08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[DRL06] Angelo Duarte, Dolores Rexachs, and Emilio Luque. Increasing the cluster availability using radic. In *CLUSTER*, 2006.

[DRL07] Angelo Duarte, Dolores Rexachs, and Emilio Luque. Functional tests of the radic fault tolerance architecture. In *PDP*, pages 278–287, 2007.

[HMS+09] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama. Warpp: a toolkit for simulating high-performance parallel scientific codes. In *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[MR09] Cyriel Minkenberg and Germán Rodriguez. Trace-driven co-simulation of high-performance computing systems using omnet++. In *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[MSSD00] Xavier Molero, Federico Silla, Vicente Santonja, and José Duato. Modeling and simulation of storage area networks. In *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 307, Washington, DC, USA, 2000. IEEE Computer Society.

[NFG+08] Alberto Nuñez, Javier Fernandez, Jose D. Garcia, Laura Prada, and Jesús Carretero. Simcan: a simulator framework for computer architectures and storage networks. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[SDRL08] Guna Santos, Angelo Duarte, Dolores Rexachs, and Emilio Luque. Providing non-stop service for message-passing based parallel applications with radic. In *Euro-Par*, pages 58–67, 2008.

[SG07] Bianca Schroeder and Garth A. Gibson. Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 78:012022 (11pp), 2007.

[TLCS09] Mustafa M. Tikir, Michael A. Laurenzano, Laura Carrington, and Allan Snavely. Psins: An open source event tracer and execution simulator for mpi applications. In *Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, pages 135–148, Berlin, Heidelberg, 2009. Springer-Verlag.