novática
CEPIS UPGRADE Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software). ***Novática*** co-edita asimismo **UP**GRADE, revista digital de **CEPIS** (*Council of European Professional Informatics Societies*), en lengua inglesa, y es miembro fundador de **UP**ENET (***UP****GRADE* **E***uropean* **NET***work*).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>
<http://www.cepis.org/upgrade>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Consejo Editorial
Ignacio Agulló Sousa, Guillem Alsina González, María José Escalona Cuaresma, Rafael Fernández Calvo (presidente del Consejo), Jaime Fernández Martínez, Luís Fernández Sanz, Didac Lopez Viñas, Celestino Martín Alonso, José Onofre Montesa Andrés, Francesc Noguera Puig, Ignacio Pérez Martínez, Andrés Pérez Payeras, Viktu Pons i Colomer, Juan Carlos Vigo López

Coordinación Editorial
Llorenç Pagés Casas <pages@ati.es>
Composición y autoedición
Jorge Llácer Gil de Ramales
Traducciones
Grupo de Lengua e Informática de ATI <http://www.ati.es/gt/lengua-informatica/>
Administración
Tomás Brunete, María José Fernández, Enric Camarero, Felicidad López

Secciones Técnicas - Coordinadores
Acceso y recuperación de la información
José María Gómez Hidalgo (Optenet), <jmgomezh@yahoo.es>
Manuel J. Maña López (Universidad de Huelva), <manuel.mana@diesia.uhu.es>
Administración Pública electrónica
Francisco López Crespo (MAE), <flc@ati.es>
Arquitecturas
Enrique F. Torres Moreno (Universidad de Zaragoza), <enrique.torres@unizar.es>
Jordi Tubella Morgadas (DAC-UPC), <jordit@ac.upc.es>
Auditoría SITIC
Marina Touriño Troitiño, <marinatourino@marinatourino.com>
Manuel Palao García-Suelto (ATI), <manuel@palao.com>
Derecho y tecnologías
Isabel Hernando Collazos (Fac. Derecho de Donostia, UPV), <isabel.hernando@ehu.es>
Elena Davara Fernández de Marcos (Davara & Davara), <edavara@davara.com>
Enseñanza Universitaria de la Informática
Cristóbal Pareja Flores (DSIP-UCM), <cpareja@sip.ucm.es>
J. Ángel Velázquez Iturbide (DLSI I, URJC), angel.velazquez@urjc.es>
Entorno digital personal
Andrés Marín López (Univ. Carlos III), <amarin@it.uc3m.es>
Diego Gachet Páez (Universidad Europea de Madrid), <gachet@uem.es>
Estándares Web
Encarna Quesada Ruiz (Virati), <encarna.quesada@virati.com>
José Carlos del Arco Prieto (TCP Sistemas e Ingeniería), <jcarco@gmail.com>
Gestión del Conocimiento
Joan Baiget Solé (Cap Gemini Ernst & Young), <joan.baiget@ati.es>
Informática y Filosofía
José Angel Olivas Varela (Escuela Superior de Informática, UCLM), <joseangel.olivas@uclm.es>
Karim Gherab Martín (Harvard University), <kgherab@gmail.com>
Informática Gráfica
Miguel Chover Sellés (Universitat Jaume I de Castellón), <chover@lsi.uji.es>
Roberto Vivó Hernando (Eurographics, sección española), <rvivo@dsic.upv.es>
Ingeniería del Software
Javier Dolado Cosín (DLSI-UPV), <dolado@si.ehu.es>
Daniel Rodríguez García (Universidad de Alcalá), <daniel.rodriguez@uah.es>
Inteligencia Artificial
Vicente Botti Navarro, Vicente Julián Inglada (DSIC-UPV), <{vbotti,vinglada}@dsic.upv.es>
Interacción Persona-Computador
Pedro M. Latorre Andrés (Universidad de Zaragoza, AIPO), <platorre@unizar.es>
Francisco L. Gutierrez Vela (Universidad de Granada, AIPO), <fgutierr@ugr.es>
Lengua e Informática
M. del Carmen Ugarte García (ATI), <cugarte@ati.es>
Lenguajes Informáticos
Óscar Belmonte Fernández (Univ. Jaime I de Castellón), <belfern@lsi.uji.es>
Inmaculada Coma Tatay (Univ. de Valencia), <Inmaculada.Coma@uv.es>
Lingüística computacional
Xavier Gómez Guinovart (Univ. de Vigo), <xgg@uvigo.es>
Manuel Palomar (Univ. de Alicante), <mpalomar@dlsi.ua.es>
Mundo estudiantil y jóvenes profesionales
Federico G. Mon Trotti (RITSI), <gnu.fede@gmail.com>
Mikel Salazar Peña (Area de Jovenes Profesionales, Junta de ATI Madrid), <mikeltxo_uni@yahoo.es>
Profesión informática
Rafael Fernández Calvo (ATI), <rfcalvo@ati.es>
Miquel Sárries Griñó (ATI), <msarries@ati.es>
Redes y servicios telemáticos
José Luis Marzo Lázaro (Univ. de Girona), <joseluis.marzo@udg.es>
Juan Carlos López López (UCLM), <juancarlos@uclm.es>
Robótica
José Cortés Arenas (Sopra Group), <joscorare@gmail.com>
Juan González Gómez (Universidad Carlos III), <juan@iearobotics.com
Seguridad
Javier Areitio Bertolín (Univ. de Deusto), <jareitio@deusto.es>
Javier López Muñoz (ETSI Informática-UMA), <jlm@lcc.uma.es>
Sistemas de Tiempo Real
Alejandro Alonso Muñoz, Juan Antonio de la Puente Alfaro (DIT-UPM),
<{aalonso,jpuente}@dit.upm.es>
Software Libre
Jesús M. González Barahona (Universidad Politécnica de Madrid), <israel.herraiz@upm.es>
Israel Herráiz Tabernero (UAX), <isra@herraiz.org>
Tecnología de Objetos
Jesus García Molina (DIS-UM), <jmolina@um.es>
Gustavo Rossi (LIFIA-UNLP Argentina), <gustavo@sol.info.unlp.edu.ar>
Tecnologías para la Educación
Juan Manuel Dodero Beardo (UC3M), <dodero@inf.uc3m.es>
César Pablo Córcoles Briongo (UOC), <ccorcoles@uoc.edu>.
Tecnologías y Empresa
Didac López Viñas (Universitat de Girona), <didac.lopez@ati.es>
Francisco Javier Cantais Sánchez (Indra Sistemas), <fjcantais@gmail.com>
Tendencias tecnológicas
Alonso Alvarez García (TID), <aad@tid.es>
Gabriel Martí Fuentes (Interbits), <gabi@atinet.es>
TIC y Turismo
Andrés Aguayo Maldonado, Antonio Guevara Plaza (Univ. de Málaga), <{aguayo, guevara}@lcc.uma.es>

Las opiniones expresadas por los autores son responsabilidad exclusiva de losmismos. ***Novática*** permite la reproducción, sin ánimo de lucro, de todos los artículos, a menos qué lo impida la modalidad de © o copyright elegida por el autor, debiéndose en todo caso citár su procedencia y enviar a ***Novática*** un ejemplar de la publicación.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn./fax 963330392 <secreval@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia <secregal@ati.es>
Suscripción y Ventas <http://www.ati.es/novatica/interes.html>, ATI Cataluña, ATI Madrid
Publicidad
Padilla 66, 3º, dcha., 28006 Madrid
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Resolución en marcha - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 211, mayo-junio 2011, año XXXVII

sumario

editorial
El papel de las TIC en los movimientos sociales > 02

en resumen
Inteligencia de negocios en clave de presente > 02
Llorenç Pagés Casas

Noticias de IFIP
Reunión del TC-1 (*Foundations of Computer Science*) > 03
Michael Hinchey,Karin Breitman, Joaquim Gabarró
Reunión anual del TC-10 (*Computer Systems Technology*) > 04
Juan Carlos López López

Actividades de ATI
V Edición del Premio *Novática* > 05

monografía

Business Intelligence
(En colaboración con **UP**GRADE)
Editor invitado: Jorge Fernández González
Presentación. *Business Intelligence*: analizando datos para extraer nueva información y tomar mejores decisiones > 06
Jorge Fernández González
***Business Information Visualization:* Representación de la información empresarial** > 08
Josep Lluís Cano Giner
***BI Usability*: evolución y tendencia** > 16
R. Dario Bernabeu, Mariano A. García Mattío
Factores críticos de éxito de un proyecto de *Business Intelligence* > 20
Jorge Fernández González, Enric Mayol Sarroca
Modelos de construcción de *Data Warehouses* > 26
José María Arce Argos
***Data Governance*: ¿qué?, ¿cómo?, ¿por qué?** > 30
Óscar Alonso Llombart
***Business Intelligence* y pensamiento sistémico** > 35
Carlos Luis Gómez
Caso de estudio: Estrategia BI en una ONG > 39
Diego Arenas Contreras

secciones técnicas

Arquitecturas
Extensiones al núcleo de Linux para reducir los efectos del envejecimiento del software > 43
Ariel Sabiguero, Andrés Aguirre, Fabricio González, Daniel Pedraja, Agustín Van Rompaey

Derecho y tecnologías
La protección de datos personales en el desarrollo de software > 50
Edmundo Sáez Peña

Enseñanza Universitaria de la Informática
Reorganización de las prácticas de compiladores para mejorar el aprendizaje de los estudiantes > 56
Jaime Urquiza Fuentes, Francisco J. Almeida Martínez, Antonio Pérez Carrasco

Estándares Web
Especificación y prueba de requisitos de recuperabilidad en transacciones WS-BusinessActivity > 61
Rubén Casado Tejedor, Javier Tuya González, Muhammad Younas
Referencias autorizadas > 70

sociedad de la información

Informática práctica
Criptoanálisis mediante algoritmos genéticos de una comunicación cifrada en la Guerra Civil > 71
Tomás F. Tornadijo Rodríguez

Programar es crear
El problema del decodificador (Competencia UTN-FRC 2010, problema C, enunciado) > 75
Julio Javier Castillo, Diego Javier Serrano
Triangulo de Pascal y la Potencia Binomial (Competencia UTN-FRC 2010, problema E, solución) > 76
Julio Javier Castillo, Diego Javier Serrano, Marina Elizabeth Cardenas

asuntos interiores

Coordinación editorial / Programación de Novática / Socios Institucionales > 77

Tema del próximo número: "*Innovación y emprendimiento en Informática*"

Extensiones al núcleo de Linux para reducir los efectos del envejecimiento del software

Ariel Sabiguero, Andrés Aguirre, Fabricio González, Daniel Pedraja, Agustín Van Rompaey

Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Montevideo (Uruguay)

<{asabigue,aaguirre}@fing.edu.uy>;
<{fabgonz,danigpc,fenix.uy}@gmail.com>

Este artículo ha sido seleccionado de entre las mejores ponencias presentadas en la XXXVI Conferencia Latinoamericana de Informática (XXXVI CLEI), evento anual promovido por el Centro Latinoamericano de Estudios en Informática (CLEI), entidad en la cual ATI es el representante de España.

Resumen: *Las técnicas actuales de rejuvenecimiento de software atacan diferentes errores transitorios de los sistemas de una forma poco específica: restauración del proceso o sistema completo. Nuestro equipo implementó un prototipo que ataca el problema con una aproximación más fina. El prototipo implementado detecta y corrige la corrupción de páginas de memoria de solo-lectura en un sistema GNU/Linux. El presente trabajo describe la clase de errores considerados, el mecanismo de corrección de errores implementado y su aplicación a la confiabilidad y disponibilidad del sistema. Para evaluar la implementación, realizamos estudios en diferentes contextos de aplicación bajo cargas de trabajo simuladas. Mecanismos de inyección de fallas fueron utilizados para generar y simular errores en el sistema. Los resultados relativos a desempeño y mejora en la disponibilidad se presentan también, mostrando que la utilización de esta técnica es adecuada en condiciones generales de un sistema.*

Palabras clave*: rejuvenecimiento, soft-errors, técnicas de corrección de memoria.*

1. Introducción

Algunas de las razones por las que las imágenes del software en ejecución se corrompen son las pérdidas de memoria, bloqueos infinitos, bloqueos en memoria compartida, hilos sin finalizar, fragmentación del almacenamiento, corrupción de los datos, interferencia por rayos cósmicos, fallos en la disipación térmica, entre otros. Este fenómeno es conocido como *envejecimiento del software*[1] , y ha sido observado en *clusters* de procesamiento, sistemas de telecomunicaciones, servidores web, PC domésticos y otros sistemas. Las evidencias de sus efectos son más comunes en sistemas que operan de forma ininterrumpida, como servidores web y de correo electrónico, entre otros.

Una forma ingenua, intuitiva y común para combatir la degradación de la imagen de software en ejecución es el reinicio periódico y preventivo de los sistemas. A esta técnica se la refiere como *rejuvenecimiento del software*[2] . La técnica trae al centro de cómputos la receta *amateur* de apagar y encender un dispositivo que no responde. Diferentes técnicas de rejuvenecimiento sugieren estrategias para los re-inicios preventivos, brindándonos la sensación de que estamos combatiendo el fenómeno desde un punto de vista tecnológico.

El presente trabajo analiza y aísla fuentes específicas de degradación de la imagen del software en memoria y su efecto en la ejecución de sus respectivos programas. Un profundo análisis de las técnicas de manejo de memoria del sistema operativo nos permite proponer técnicas de rejuvenecimiento que no requieren el reinicio del sistema. Se presentan, asimismo, los resultados de la implementación de un prototipo, así como los datos experimentales.

La organización del trabajo se describe a continuación. La **sección** 2 provee una introducción y análisis del fenómeno del envejecimiento del software. La **sección** 3 brinda una introducción al manejo de memoria en los sistemas operativos modernos, poniendo foco en el manejo de la memoria de solo-lectura y las técnicas de detección de corrupción. Posteriormente, la **sección 4** brinda detalles del prototipo implementado, junto con resultados experimentales relevantes. Diferentes líneas de trabajo para el área son sugeridas en la **sección 5**. El trabajo concluye en la **sección 6**.

2. Envejecimiento del software

No todas las fallas en nuestros sistemas se deben a software mal construido. De hecho, podríamos tener software perfecto ejecutándose en un sistema, y dicho sistema podría fallar debido a causas externas. Existe una alta probabilidad de que en muchos de estos fallos, causas externas afecten el hardware del sistema, y por consiguiente al software que allí se ejecuta. También existe la posibilidad de que ciertas condiciones de hardware o software transitorias afecten la ejecución esperada. El resto de esta sección introduce conceptos relevantes relacionados al envejecimiento de software.

2.1. Taxonomía

Jim Gray [1] clasifica los errores de software en dos categorías distintas, Bohrbugs y Heisenbugs. Esta distinción se basa en la facilidad de reproducir la falla causada por estos errores.

La primera categoría, los Bohrbugs, que deben su nombre al modelo atómico de Bohr, son esencialmente errores permanentes en el diseño, y son por naturaleza casi deterministas. Pueden ser identificados y corregidos durante la fase de pruebas del ciclo de vida del software.

Por otro lado, los Heisenbugs, así nombrados por el principio de incertidumbre de Heisenberg, incluyen aquellas fallas internas que son intermitentes. Fallas cuyas condiciones de activación ocurren raramente, y por lo tanto son difíciles de reproducir. Un manejo de excepciones incorrecto o incompleto es un claro ejemplo de Heisenbugs. Estos errores son más difíciles de detectar durante la fase de pruebas que los Bohrbugs.

K. Vaidyanathan y Kishor S. Trivedi [2] agregan una tercera categoría a esta clasificación, la cual incluye aquellas fallas causadas por la ocurrencia de envejecimiento de software.

Esta categoría de fallas es similar en ciertos aspectos a los Heisenbugs, ya que su activación está determinada por ciertas condiciones como la falta de recursos del sistema operativo, algo que no es sencillo de reproducir.

Sin embargo, sus métodos de recuperación difieren significativamente. Mientras que los Heisenbugs se corrigen mediante técnicas puramente reactivas, las fallas causadas por el envejecimiento de software pueden ser prevenidas mediante la aplicación de técnicas proactivas, como puede ser el reinicio periódico del software afectado.

Estamos particularmente interesados en los errores de hardware, especialmente aquellos que ocurren en los chips de memoria de un sistema. Estos errores también son conocidos como *soft errors* [3] y pueden ser clasificados como errores causados por envejecimiento de software, aunque no son causados por el envejecimiento en sí mismo, sino por causas externas, como pueden ser los rayos cósmicos, temperatura y humedad, entre otros.

2.2. Soft Errors

Como se mencionó en la **sección 2.1.**, los

soft errors son errores en el hardware de memoria de un sistema. Es importante enfocarse en ellos, ya que normalmente no se toman en cuenta al momento de diseñar software, y pueden eventualmente causar fallas totales en un sistema.

Sería interesante contar con algún tipo de protección contra estos errores, que no esté construida directamente sobre el hardware, como los chips de memoria con código corrector de errores (memoria ECC).

Debido a la naturaleza de estos errores, es imposible predecir cuándo ocurrirán, y por lo tanto, necesitamos una técnica reactiva que permita al software recuperarse frente a la aparición de uno de ellos.

Los *soft errors* pueden ser causados por una variedad de eventos. James F. Ziegler, un investigador de IBM, publicó varios artículos probando que los rayos cósmicos podían ser otra causa de aparición de *soft errors* [4].

Durante estos experimentos, IBM descubrió que la frecuencia de aparición de *soft errors* se incrementaba con la altura, duplicándose a 800 metros por encima del nivel del mar. En la ciudad de Denver, a 1.600 metros por encima del nivel del mar, la frecuencia de aparición de *soft errors* es 10 veces la del nivel del mar.

En otro experimento, descubrieron que los sistemas ubicados bajo tierra experimentaban una frecuencia de *soft errors* mucho menor, y dado que 20 metros de roca pueden bloquear casi la totalidad de rayos cósmicos, concluyeron que dichos rayos cósmicos eran efectivamente una posible causa de aparición de *soft errors*.

Otra posible causa de aparición de *soft errors* es la interferencia electromagnética (EMI). Esta interacción puede causar alteraciones en los transistores o buses y, con el tiempo, puede causar que su valor lógico cambie. Este tipo de interferencia es común en lugares donde operan motores eléctricos de alta frecuencia [5].

En entornos industriales donde se utilizan este tipo de motores y muchos sistemas empotrados controlan la maquinaria, la probabilidad de ocurrencia de un *soft error* es mucho más alta, lo cual puede llevar a daños en la maquinaria o peor aún, lesiones en los operarios humanos. Estos factores son cruciales a la hora de considerar aspectos de seguridad.

2.3. Soluciones y técnicas existentes

Las técnicas más comunes para combatir los errores causados por el envejecimiento de software se conocen como "técnicas de rejuvenecimiento de software".

Existen dos estrategias distintas que pueden utilizarse para determinar el momento adecuado para aplicar el rejuvenecimiento.

La primera es conocida como enfoque *open-loop* [2]. Este método consiste en aplicar el proceso de rejuvenecimiento sin utilizar ninguna clase de información sobre el desempeño del sistema. El rejuvenecimiento se puede aplicar luego de que una cierta cantidad de tiempo ha pasado desde la última aplicación, o cuando el número de trabajos concurrentes del sistema llega a un cierto número crítico.

Por otro lado, tenemos el enfoque *closed-loop* [2]. En este método, la salud del sistema es continuamente monitoreada para detectar la posibilidad de que ocurra un error causado por el envejecimiento, como la falta de un recurso particular del sistema, que pueda llevar a una disminución del rendimiento o a una falla total del sistema. En este enfoque podemos clasificar nuestras técnicas según la forma de analizar los datos recolectados.

El *análisis offline* utiliza datos sobre el rendimiento del sistema, recolectados durante un extenso período de tiempo (semanas o meses) para determinar la frecuencia óptima de ejecución del rejuvenecimiento, siendo adecuado para sistemas cuyo comportamiento es determinístico.

El *análisis online* se basa en juegos de datos recolectados mientras el sistema está en funcionamiento, que se combinan con juegos de datos anteriores, para decidir si es un buen momento para aplicar rejuvenecimiento, siendo aplicable para sistemas cuyo comportamiento es difícil de predecir.

Las técnicas de rejuvenecimiento de software se pueden aplicar en distintos niveles de granularidad.

Un ejemplo de rejuvenecimiento aplicado a nivel de sistema es un reinicio total de hardware, y a nivel de aplicación el reinicio de procesos afectados es otro ejemplo de rejuvenecimiento.

La aplicación de rejuvenecimiento tiene un impacto negativo en la disponibilidad del sistema rejuvenecido. Por este motivo es muy útil contar con una arquitectura de alta disponibilidad, donde podemos aplicar rejuvenecimiento a ciertos nodos individuales sin impactar en la disponibilidad del sistema globalmente.

Si diseñamos nuestros sistemas teniendo en cuenta todos estos factores, aplicar rejuvenecimiento ha probado ser una técnica proactiva muy efectiva y de muy bajo costo [6] para prevenir errores. Con este enfoque no estamos mejorando la confiabilidad del sistema pero sí la disponibilidad de los servicios.

3. Considerando los *soft errors* y la memoria de solo lectura

Las arquitecturas de hardware y sistemas operativos modernos proveen mecanismos de protección sobre la memoria disponible, esencialmente, asignando privilegios de acceso a cada una de las páginas de memoria virtual. La asignación de privilegios de acceso a las páginas de memoria está directamente soportada por la mayoría de las arquitecturas, debido al soporte provisto por los microprocesadores actuales.

A lo largo del presente trabajo nos referimos con el término *memoria de solo-lectura (RO por sus iniciales en inglés)* a las partes de la memoria de un sistema a las cuales los mecanismos de protección del sistema operativo prohíben la modificación de su contenido.

De las múltiples facilidades ofrecidas por el sistema de memoria, cabe destacar que en cualquier instante del tiempo podemos clasificar las páginas de un proceso en dos categorías: solo-lectura y modificables. Las páginas de solo-lectura son constantes a lo largo de la ejecución de un programa.

En base a la definición anterior de memoria de solo-lectura, es directo ver como los cambios observados en dichas regiones son interpretados como una falla.

Asumiendo que el mecanismo de protección brindado por el sistema operativo está carente de errores, el único motivo por el que podemos encontrar un cambio en la memoria de solo lectura es debido a un *soft error*. Bajo esta hipótesis asuminos que el uso de técnicas de detección de cambios en memoria de solo-lectura es equivalente a la detección de *soft errors*.

3.1. Memoria RO en software estándar

Cualquier programa corriendo en un sistema operativo estándar actual está conformado por un conjunto de archivos-objeto, conteniendo bibliotecas y ejecutables. Compiladores y enlazadores trabajan en conjunto con el SO para convertir código fuente en código objeto que puede ser cargado y ejecutado en su plataforma objetivo. Existen muchos formatos diferentes para archivos ejecutables, como ELF (*Executable and Linking Format*), normalmente utilizado en sistemas GNU/Linux, o PE (*Portable Executable*) utilizado en sistemas basados en Windows NT.

La evolución en los lenguajes de programación, verificaciones y restricciones impuestas por los nuevos compiladores, están convirtiendo el código ejecutable en un código más seguro. Los ejecutables construidos con compiladores actuales fuerzan la separación del código objeto en páginas de solo-lectura y otras con privilegios de escritura.

La proporción entre éstas es dependiente de los programas particulares en sí mismos. Programas compactos que manipulan grandes volúmenes de datos (p.e. software de *streaming*), presentan una cantidad reducida de páginas de memoria de solo-lectura. Programas de alta complejidad que manipulan juegos de datos pequeños (p.e. paquetes de oficina) presentan una tasa mucho mayor de páginas de solo-lectura. Es por esta razón que la cobertura ofrecida por la presente técnica no es la misma en todos los escenarios. Algunos resultados experimentales son presentados en la **sección 4.3.**

3.2. Memoria RO en Linux

El formato estándar de objetos, ejecutables y bibliotecas en GNU/Linux es el formato ELF.

ELF soporta el concepto de secciones, que son colecciones de información de tipos similares. Cada sección representa una porción del archivo. A modo de ejemplo mencionamos que el código ejecutable es colocado en una sección conocida como .text. Las variables inicializadas por el usuario son colocadas en una sección denominada .data y los datos sin inicializar en la sección .bss.

El gestor de memoria puede marcar porciones de memoria como RO de forma tal que los intentos de modificación de las mismas resultan en la finalización de la ejecución del programa por parte del sistema operativo. Es por ello que, en vez de no esperar que una posición de memoria RO cambie como consecuencia de un error en el programa, tenemos la certeza de que cualquier intento de modificación de una posición RO es un error fatal, indicando un fallo en el software. El sistema operativo se encarga de finalizar el programa que ofendió la protección.

Dado que deseamos que las porciones ejecutables estén en páginas de memoria RO y las posiciones de memoria modificables en memoria de lectura-escritura, resulta más eficiente agrupar todas las porciones ejecutables en la sección .text, y todas las áreas que pueden ser modificadas en la sección .data. Los datos experimentales relativos a un servidor web estándar (LAMP) son presentados en las **secciones 4.2. y 4.3**.

Los sistemas empotrados representan otro entorno de aplicación en el que GNU/Linux está ganando terreno. Estos sistemas empotrados corren generalmente en plataformas en las que no se dispone de mecanismos de detección y corrección de errores en memoria.

Los beneficios de una solución estándar, basada en servicios del sistema operativo, son de gran interés en estos sistemas que corren Linux, en los que las fallas tienen un impacto directo en características como la confiabilidad de los mismos.

4. Implementación de un prototipo en GNU/Linux

Describiremos la implementación de un prototipo que hace uso de la hipótesis presentada en la **sección 3** para combatir los efectos del envejecimiento en el software causado por errores en la memoria. La herramienta se desarrolló como una extensión al núcleo de GNU/Linux, extendiendo la funcionalidad del subsistema de memoria (*Memory Manager*).

En la **sección 4.1** describimos la funcionalidad del prototipo y los detalles más relevantes de su implementación. Dado que su función se basa en la detección de *soft errors*, las pruebas de validación requirieron la alteración de la memoria, de forma que puedieran ser consideradas equivalentes a los errores que buscamos detectar. Como se describe en la **sección 4.2**, fue necesario hacer uso de técnicas de inyección de fallos, lo que nos permitió simular errores de hardware de una forma repetible, económica y segura. Por último, en la **sección 4.3** presentamos los resultados de algunas pruebas funcionales y de desempeño del prototipo ejecutándose en una instancia concreta de GNU/Linux. Las pruebas de desempeño se basan en pruebas estándar de algunos aplicativos en Linux.

4.1. Detalles de la implementación

Como parte del trabajo de tesis de González, Pedraja y Van Rompaey [7], se construyó un prototipo basado en Linux. La primera meta de este prototipo fue detectar la ocurrencia de *soft errors* en la memoria física del sistema, sacando provecho de la existencia de regiones de memoria de solo lectura.

Como se vio en la **sección 3**, Linux administra los recursos de memoria con granularidad de páginas, por lo que el primer paso para lograr el objetivo mencionado es identificar el subconjunto páginas para los que el S.O. nos garantiza acceso de solo lectura. Este esquema de protección es representado por medio de los permisos de acceso de las áreas de memoria virtual que forman parte del espacio de direcciones de un proceso y se hace cumplir por medio de las tablas de páginas del proceso. El mecanismo de memoria virtual de Linux hace posible que las páginas de memoria sean compartidas entre los espacios de memoria de varios procesos con diferentes permisos de acceso en cada caso. Tomando esto en cuenta, se puede definir el subconjunto de páginas de solo-lectura como aquellas que están asignadas al espacio de direcciones de uno o más procesos del espacio de usuario, con permisos de solo-lectura en todos los casos.

Todos los eventos de cambios en la asignación, y permisos de acceso de las páginas de memoria, fueron analizados. Manejando estos eventos y manteniendo información de estado extra, fue posible determinar en cualquier momento la pertenencia o no de una página dada al subconjunto deseado. Los cambios realizados al núcleo se limitaron lo más posible y la mayoría de la funcionalidad fue encapsulada e implementada en un módulo del núcleo (ver **figura 1**).

Una vez identificado el subconjunto de páginas de solo-lectura, el segundo paso es contar con un mecanismo de detección de cambios a nivel de bits sobre sus elementos. Cualquier cambio en un bit en este contexto se puede interpretar como la ocurrencia de un *soft error*.

La solución general para proveer esta clase de funcionalidad es mantener, para cada página en el conjunto, un código de redundancia sobre los bytes pertenecientes al rango de memoria que este representa. En el prototipo Linux, esto fue implementado agregando un código de redundancia al estado mantenido para cada página de memoria. El código tenía como requerimiento proveer detección de errores en múltiples bits de una forma eficiente, es decir, usando una pequeña cantidad de bits de redundancia para los volúmenes de datos objetivo (el tamaño de una página de memoria va desde 4KB a 4MB).

Luego de evaluar diferentes opciones, el algoritmo de detección de errores seleccionado para esta tarea fue CRC32. La redundancia mantenida debe ser actualizada en los eventos mencionados anteriormente, de forma que sea la correspondiente a los contenidos de la página mientras la misma pertenece al conjunto de solo-lectura. La estrategia seguida para garantizar esto consiste en recalcular y almacenar el código cada vez que la página ingresa a un estado que lo posiciona dentro del conjunto (por ejemplo cuando la primer asignación de solo-lectura es realizada sobre una página que previamente estaba libre), y borrar (o ignorar) la redundancia en transiciones a estados fuera del conjunto (como puede ser cuándo una asignación de escritura es agregada a una página que era previamente de solo-lectura). Gracias al estado almacenado, verificar la existencia de errores es tan simple como recalcular el CRC32 y compararlo con el almacenado.

La herramienta puede ser configurada para seguir una de tres estrategias de búsqueda de errores. Para comprender las diferentes estrategias, es importante tener en cuenta que, debido a que estamos tratando de reaccionar a eventos que ya habrán sucedido, la detección de errores será asíncrona a la ocurrencia de los mismos y siempre habrá un retraso entre ambos sucesos. Hacer ese intervalo de tiempo lo más pequeño posible fue la meta al seleccionar las estrategias de búsqueda, puesto que esto reduce las probabilidades de que un proceso acceda a la sección de memoria

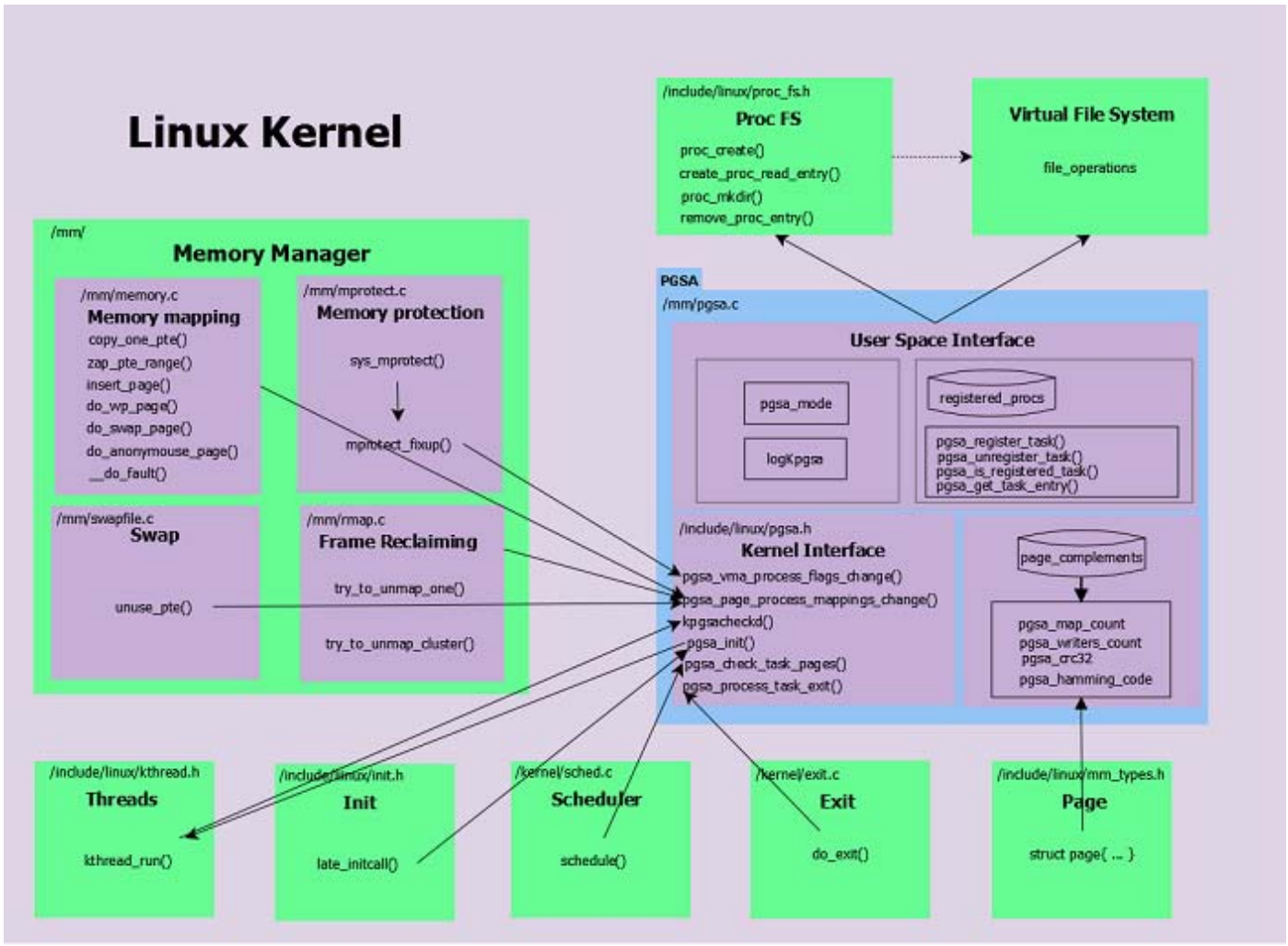

Figura 1. Cambios realizados al núcleo de Linux.

corrupta antes de que se puedan tomar acciones.

El primer y más simple de los enfoques implementado consiste en un hilo del núcleo que chequea continuamente todo el conjunto de páginas de solo-lectura en un ciclo constante. Aunque esta estrategia puede sugerir una pérdida de desempeño, en la práctica ha probado ser bastante efectiva.

El prototipo brinda algunas funcionalidades de configuración, las que permiten a un administrador del espacio de usuarios ajustarla de acuerdo a cada escenario y sus requerimientos. Trataremos sobre las mismas luego, pero se da aquí un adelanto sobre el registro de procesos, que es esencial para la existencia de las dos estrategias de búsqueda restantes. El registro de procesos básicamente permite al administrador configurar la herramienta de forma que ésta enfoque sus esfuerzos en un grupo indicado de procesos.

Cuando el módulo es configurado para trabajar solo con tareas registradas, puede realizar verificaciones de errores de dos maneras. La primera es similar a la estrategia ya explicada, solo que en este caso el hilo itera sobre la lista de tareas registradas chequeando solamente aquellas páginas de solo lectura asignadas al espacio de memoria de la tarea actual. Esta estrategia ha probado ser considerablemente más eficiente que la anterior.

Los experimentos realizados para comparar los retrasos de detección de errores entre ambas técnicas mostraron un intervalo de tiempo promedio de 275 milisegundos para la básica. Por otro lado, las pruebas indicaron un retraso promedio de 1 milisegundo para la estrategia que usa los procesos registrados, aún cuando se estaba simulando carga al sistema al registrar las tareas del *stack* LAMP.

La última estrategia de búsqueda de errores implementada también trabaja solo con las tareas registradas. La misma fue desarrollada modificando el planificador de tareas del núcleo, para chequear las páginas de solo-lectura asignadas al espacio de memoria de la próxima tarea a obtener la UCP, justo antes de que realmente obtenga el recurso. La meta de esta estrategia es permitir la toma de acciones en todas las secciones de memoria corruptas pertenecientes al espacio de memoria de un proceso, antes de que el mismo obtenga el control de la UCP y logre acceder a dichas secciones. A diferencia de los dos enfoques presentados previamente, en este caso el objetivo no es reducir el retraso de detección de los errores. Esta estrategia ha probado ser efectiva, pero puede también significar una importante sobrecarga en la planificación de procesos, por lo que debería ser usada con extrema cautela.

El solo hecho de la detección no es suficiente, por lo que se definió una secuencia de acciones para el manejo de errores como parte del prototipo. El primer paso consiste en emplear un código corrector de errores, para tratar de arreglar la memoria afectada de forma automática. Existen varios tipos de códigos ampliamente utilizados para este tipo de tarea, entre los cuales se seleccionó una adaptación de los *Códigos de Hamming*, debido a su fácil implementación. Los Códigos de Hamming son en realidad el método más comúnmente usado por los chips de memoria ECC, pero en este caso la corrección de un solo bit que proveen puede no ser suficiente para toda una página de memoria. Con diferentes características de cubrimiento y desempeño, las técnicas Reed-Solomon podrían haberse aplicado, pero las mismas fueron descartadas debido a la complejidad de su implementación.

La segunda acción tomada para tratar de corregir errores se basó en el concepto de rejuvenecimiento, concebido como técnica pro-activa de manejo de fallos, orientada a limpiar el estado interno del sistema para prevenir la ocurrencia de fallos más severos que provoquen caídas en el futuro. Una posible solución para corregir páginas de memoria de solo-lectura corruptas usando rejuvenecimiento, es mantener un respaldo de sus contenidos y recargar el mismo ante errores. La mayoría de las páginas de solo-lectura son la imagen de un archivo en disco, por lo que el propio archivo se convierte en el respaldo requerido para estas páginas. El prototipo provee una funcionalidad que recarga automáticamente la imagen de una página desde disco, cuando se detecta un *soft error* dentro del mismo. Para el caso de páginas no vinculadas a archivos (también llamadas anónimas) se debería proveer una implementación de almacén de respaldo, pero su desarrollo excedió el alcance del prototipo.

Se encontró una forma de aplicar rejuvenecimiento cuando se ejecuta el núcleo, pero en realidad la mayoría de la teoría sobre esta técnica apunta al espacio de usuario. Dado que muchas estrategias de rejuvenecimiento se basan en estadísticas de los recursos para decidir cuándo aplicarlo, se entendió que en este caso el rol del SO era proveer información sobre la detección de errores y notificaciones, de forma que asista a esas decisiones.

Primero, se decidió brindar notificaciones sincrónicas de los errores detectados a los agentes de rejuvenecimiento del espacio de usuario. Esta funcionalidad se implementó por medio de señales de Linux. Cuando un proceso es registrado en la herramienta, se especifica también otro proceso que toma el rol de agente de rejuvenecimiento de la tarea registrada. En caso de detección de errores en el espacio de memoria de la tarea, una señal específica es enviada a su agente para que éste pueda tomar acciones.

A veces el rejuvenecimiento simplemente consiste en el reinicio de procesos o incluso de todo el sistema. Sin embargo, si se cuenta con la información adecuada, se pueden tomar acciones de mayor granularidad (por ejemplo, volver a asignar solo un archivo). Para brindar a los agentes la oportunidad de reconocer tales acciones, el prototipo publica al espacio de usuario información detallada sobre cada error detectado. Estos detalles incluyen para cada proceso (registrado o no), las direcciones física y virtual exactas de los bytes cambiados, el área virtual de memoria a la cual pertenece la dirección en el espacio de direcciones del proceso y a qué tipo de vínculo (archivo, anónimo) corresponde. En caso de que el error ocurra en la asignación de un archivo, se provee también el nombre del archivo y el rango asignado.

Finalmente, el estado resultante del error (corregido o no) también se muestra. La interfaz con el espacio de usuarios seleccionada para presentar esta información fue sistema de archivos virtual */proc*, usando carpetas y archivos a nivel de tareas.

Se mencionó que el prototipo también provee algunas funcionalidades de configuración, como los ya explicados procesos registrados. El módulo soporta además el concepto de modo global el cual está compuesto por un conjunto de banderas. Cada una de estas banderas permite al administrador activar o desactivar diferentes funcionalidades. Se crearon banderas adicionales para poder elegir entre las posibles estrategias de búsqueda de errores y para encender y apagar las acciones de manejo de errores o incluso todo el módulo. La interfaz con el espacio de usuarios para la configuración del modo y las tareas registradas fue implementada también usando el sistema virtual de archivos */proc*, mediante la escritura en archivos especiales.

Se ha usado el término *módulo* a lo largo de esta sección para referirse a la implementación del prototipo, pero no se ha explicado como la misma se integra realmente con el núcleo de Linux.

Como el término lo implica, todas las funcionalidades presentadas se agrupan en un único módulo en el núcleo. El mismo provee un grupo de interfaces con un conjunto de *callbacks* que le permiten manejar los eventos internos del núcleo y las interfaces con el espacio de usuarios. Estas *callbacks* se diseñaron para reducir lo más posible el acoplamiento del núcleo con el prototipo. Cuando se dice módulo, no se refiere al concepto de módulo dinámico que soporta el núcleo. Dichos módulos son capaces de agregar funcionalidad a una instancia activa del núcleo y son mayormente usados para desarrollar controladores de dispositivos. En este caso, debido a las secciones del núcleo donde las interfaces del módulo son invocadas, necesitamos agregar funcionalidad al núcleo en tiempo de compilación. Debido al tipo de funcionalidad que provee, se decidió incluir los fuentes del módulo como parte del subsistema de manejo de memoria del núcleo de Linux.

4.2. Pruebas funcionales del prototipo

La verificación de la correctitud del prototipo fue importante por varias razones. Por un lado, siempre está la necesidad de agregar calidad a la implementación que está siendo validada, pero, por otra parte, se requería separar los defectos del prototipo de los *soft errors* en la memoria. Los tests consistieron en alterar valores de memoria localizados en páginas de solo lectura y verificar que los cambios fueron detectados y corregidos.

Para el diseño de los casos de prueba no se contaba con medios para inyectar errores por hardware en los subsistemas de memoria, por lo que se recurrió a la inyección de errores por software. Los detalles tecnológicos implicados están fuera del alcance de este trabajo, pero simplemente podemos mencionar que una API de alto nivel del núcleo no puede ser usada para modificar las páginas de memoria de solo-lectura. Alterar los valores de memoria de solo-lectura desde el espacio de usuarios genera fallos de segmentación. Alterar los valores de memoria de solo-lectura desde una API portable de alto nivel (como *ptrace*) genera efectos no deseados; el núcleo convierte la página de solo-lectura en una de lectura-escritura antes de efectivizar el cambio. Tan pronto como la página se convierte en una de lectura-escritura, es removida del conjunto de páginas monitoreadas por el prototipo.

La inyección de errores por software debe ser hecha en modo supervisor, donde se obtiene acceso sin restricciones a la memoria y las modificaciones directas a la misma pueden saltear el esquema de protección de memoria del núcleo. Se implementó la inyección direccionando la memoria directamente desde el espacio virtual del núcleo, evitando las limitaciones de las tablas de páginas de los procesos de espacio de usuarios. Se expuso esta funcionalidad al espacio de usuarios por medio de una nueva llamada al sistema (nombrada sys_kpgsa_inject()) y se desarrolló una variedad de herramientas de alto nivel. Las opciones incluyen desde alterar posiciones de memoria de un solo byte, hasta simular fallos de memoria periódicos y aleatorios (lluvia de rayos cósmicos).

4.3. Pruebas de desempeño

Un sistema con un buen rendimiento es importante, a pesar de las positivas consecuencias sobre la confiablidad, disponibilidad y seguridad física (*safety*) de las técnicas de rejuvenecimiento automático y de software de propósito general. Con el fin de monitorear los cambios, la memoria de solo-lectura del sistema debe ser explorada completamente de forma periódica, transformando el prototipo en una aplicación con uso intensivo de UCP, sin impacto en el uso de entrada/salida. Esto introduce competencia por el acceso a memoria, y además, pérdida en la localidad.

Se analizaron dos escenarios diferentes de aplicaciones: aplicaciones con uso intensivo de UCP y las de uso intensivo de entrada/salida. Como aplicación intensiva en uso de UCP se seleccionó POV-Ray [8] (ver **figura 2**), y la de uso intensivo en entrada/salida es un servidor LAMP[3].

Para medir el impacto sobre el rendimiento del sistema utilizamos las propias herramientas de *benchmarking* que incluyen estos paquetes de software, comparando su rendimiento en

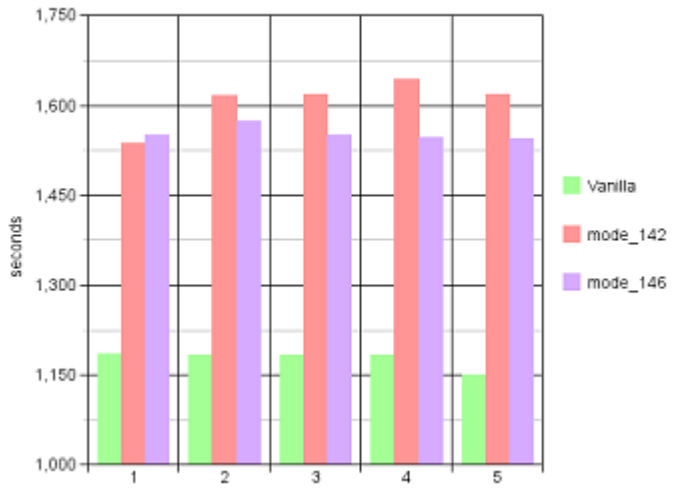

Figura 2. POV-Ray: tiempo de render.

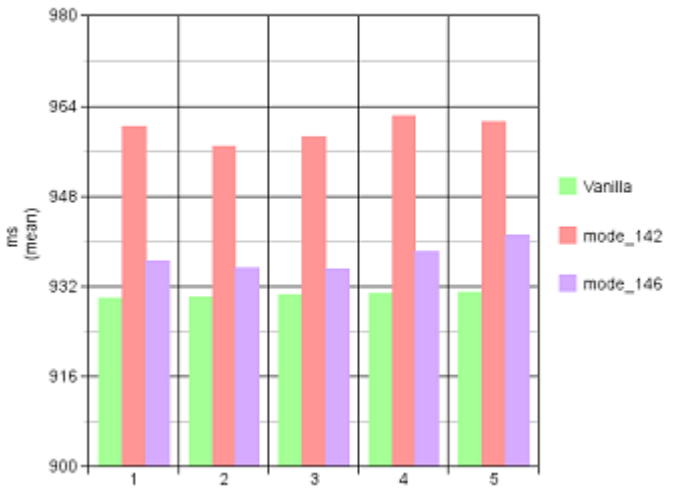

Figura 3. LAMP: tiempo de servicio.

el mismo sistema, con y sin uso de las rutinas de corrección de corrupción de memoria. Los tests fueron realizados en ambos modos.

Las mediciones de desempeño fueron consistentes con la intuición. El impacto en el rendimiento en un servidor LAMP es marginal, como puede verse en la **figura 3**, mientras el impacto en la aplicación de técnicas de *raytracing* es significativo, como se muestra en la **figura 2**.

En otras palabras, las rutinas de corrección de corrupción de memoria prácticamente no compiten con tareas que hacen un uso intensivo de IO mientras si lo hacen con las tareas que hacen uso intensivo de UCP.

Como última observación sobre el impacto en el funcionamiento del software, no todos los sistemas tienen la misma proporción de páginas de solo-lectura sobre páginas de lectura-escritura. En base a los resultados anteriores analizamos el mecanismo de funcionamiento de Apache y MySQL, y ambos comparten el paradigma de atención basado en *fork*. Los procesos padre tienen una tasa superior a la de los hijos, como se pude ver en la **tabla 1**. Los resultados observados son consistentes con el hecho que el proceso padre no almacena datos de ejecución requeridos para contestar a clientes particulares, sino información de gestión global del servicio. Los procesos que tratan con los pedidos son los clientes. Los procesos padres están en ejecución durante más tiempo que los hijos en este escenario, y son estos los procesos que más se benefician de los algoritmos de corrección de corrupción.

El impacto de un *soft error* no corregido en un proceso hijo podría afectar solamente a una única sesión, mientras que un error similar en un proceso padre podría afectar a la totalidad del servicio de allí en más. La detección de corrupción toma más relevancia en los procesos padre, donde además la tasa RO/RW es también mayor.

5. Resultados y trabajo futuro

Quedan todavía diferentes parámetros de ajuste ya identificados que deberían ser agregados a la implementación, de forma que se vuelva más adaptable a diferentes escenarios de aplicación. *freqd* y otros demonios de compensación de UCP (o *CPU equalization*) tienen que ser considerados para alcanzar también las plataformas móviles. La implementación actual mantendría el UCP de un *notebook* al 100%, consumiendo la batería demasiado rápido.

Portar el prototipo a diferentes plataformas es necesario para acceder al mercado de sistemas empotrados. Su uso en sistemas empotrados, donde comúnmente no se cuenta con memoria ECC, es un área donde el impacto puede ser significativo.

La implementación actual probó su capacidad de mejorar la disponibilidad de un servidor LAMP, bajo carga pesada y bajo lluvia pesada de rayos cósmicos simulada. Un sistema sin una implementación de corrección de corrupción de memoria de solo-lectura, falla luego de unos pocos segundos de modificaciones aleatorias de su memoria. La corrección de corrupción de memoria mostró que es posible hacer que un sistema recargue imágenes frescas del software en la RAM cuando ocurre la corrupción, sin impacto en los servicios que este provee.

Como observación final, todavía tenemos que asegurarnos que la implementación es capaz de detectar y corregir *soft errors reales*. Nuestras estimaciones [7] indican que en sistemas individuales de vanguardia (4GB DDR2 memory), habrá un error corregible, manejable para nuestro prototipo, cada 22 años en promedio por sistema. Todavía no hemos encontrado ningún error en núcleos modificados ejecutándose en ambientes de producción. Es deseable obtener acceso a ambientes donde la tasa de error es mayor.

6. Conclusiones

La industria ya sabe acerca del *software aging* o envejecimiento del software: cuando un

	MySQL				Apache httpd			
	Padre		Hijo		Padre		Hijo	
Páginas de solo lectura	287	47%	614	13%	930	38%	952	23%
Páginas de lectura/escritura	330	53%	4235	87%	1487	62%	3245	77%

Tabla 1. Tasa de páginas RO/RW en el software utilizado.

sistema comienza a tener un comportamiento errático, lo primero es reiniciarlo.

Las técnicas de rejuvenecimiento de software han ganado un lugar más significativo en la administración de sistemas en los últimos años y distintos productos las introducen, con diferentes niveles de granularidad, como parte de sus soluciones potenciadas para la disponibilidad. Hasta donde llega nuestro conocimiento, solo el rejuvenecimiento a nivel de todo el sistema o todo el proceso han sido presentados hasta el momento.

Nuestro prototipo ha probado que el rejuvenecimiento de granularidad fina es posible, como una herramienta del sistema operativo, en ambientes de aplicación de propósito general. Aunque la solución implementada solo cubre las páginas de memoria de solo lectura, la misma no requiere ningún cambio a nivel de despliegue, como es el caso de técnicas basadas en réplicas y balance de carga.

Debido al hecho de que es aplicable a cualquier pieza de software sin necesidad de realizar cambios, ofrece una forma simple y no intrusiva de reducir los costos de manejo de un ambiente de producción.

Todavía no podemos reemplazar la memoria ECC con una solución por software, pero nuestra implementación de corrección de la corrupción de memoria es capaz de mejorar la disponibilidad de los sistemas donde la tecnología ECC no está disponible.

Aunque todavía se requiere más investigación, mostramos que es posible mejorar la disponibilidad y confiabilidad de aplicaciones arbitrarias con técnicas generales de rejuvenecimiento de granularidad fina, implementadas a nivel del sistema operativo, con un impacto razonable en el desempeño.

Referencias

[1] Jim Gray. Why do computers stop and what can be done about it? *Tandem Computers. Technical Report 85.7, PN 87614,* <http://www.hpl.hp.com/techreports/tandem/TR-85.7.pdf>, 1985.

[2] Kishor S. Trivedi, Kalyanaraman Vaidyanathan, Katerina Goseva-popstojanova. Modeling and analysis of software aging and rejuvenation. *Proceedings of the IEEE Annual Simulation Symposium,* pages 270–279, 2000.

[3] Actel Corporation. Understanding soft and firm errors in semiconductor devices. *Actel Corporation. 51700002-1/12.02*, <http://www.actel.com/documents/SER_FAQ.pdf>, 2002.

[4] J. F. Ziegler. Terrestrial cosmic rays. *IBM J. Res. Dev., 40(1)*: pp.19–39, 1996.

[5] G.L. Skibinski, R.J. Kerkman, D. Schlegel. Emi emissions of modern pwm ac drives. *Industry Applications Magazine, IEEE, 5(6)*: pp. 47–80, nov/dec 1999.

[6] Artur Andrezejak, Monika Moser, Luis Silva. Managing Performance of Aging Applications Via Synchronized Replica Rejuvenation. En Alexander Clemm, Lisandro Zambenedetti, and Rolf Stadler, editors, (DSOM 2007) Managing *Virtualization of Networks and Services*, ISBN 978-3-540-75693-4, pages 98–109. Springer, 2007.

[7] Daniel Pedraja, Fabricio González, Agustín Van Rompaey. *Herramientas del sistema operativo para combatir el software aging*. Proyecto de grado, <http://www.fing.edu.uy/~asabigue/prgrado/softwareAging.pdf >.

[8] Persistence of Vision Raytracer. *POV-Ray*, <http://www.povray.org>.

Notas

[1] Del inglés *software aging*.
[2] Del inglés, *software rejuvenation*.
[3] Linux, Apache, MySQL, PHP.

JAIIO

Las **JAIIOs**, Jornadas Argentinas de Informática, se realizan desde 1961 y son organizadas por **SADIO**.

En sesiones paralelas se presentan trabajos que se publican en Anales, se discuten resultados de investigaciones y actividades sobre diferentes tópicos, desarrollándose también conferencias y reuniones con la asistencia de profesionales argentinos y extranjeros. Las JAIIOs se organizan como un conjunto de simposios separados, cada uno dedicado a un tema específico, de uno o dos días de duración, de tal forma de permitir la interacción de sus participantes.

Una lista completa de todos los simposios de las 40 JAIIO se pueden encontrar en: <http://www.40jaiio.org.ar>.

Fecha: del 29 de agosto al 02 de septiembre de 2011
Lugar: Ciudad de Córdoba, Argentina (UTN - Facultad Regional de Córdoba).