ANÁLISIS E IMPLEMENTACIÓN DE UN MECANISMO DE TOLERANCIA A FALLAS Y RESTAURACIÓN PARA UNA INFRAESTRUCTURA DE CÁLCULO DISTRIBUIDO EN REDES UNIVERSITARIAS

AUTOR
ROSEMBERG JOSÉ URIBE ESPINOSA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENERIA DE SISTEMA E INFORMATICA
BUCARAMANGA
2011

ANÁLISIS E IMPLEMENTACIÓN DE UN MECANISMO DE TOLERANCIA A FALLAS Y RESTAURACIÓN PARA UNA INFRAESTRUCTURA DE CÁLCULO DISTRIBUIDO EN REDES UNIVERSITARIAS

AUTOR
ROSEMBERG JOSÉ URIBE ESPINOSA

Trabajo para optar por el titulo de
Ingeniero de sistemas

DIRECTOR
ING. JUAN CARLOS ESCOBAR RAMIREZ

CODIRECTOR
Ph.D CARLOS JAIME BARRIOS HERNANDEZ

ASESORES
ING. ANTONIO JOSE LOBO FIGUEROA
ING. CRISTIAN CAMILO RUIZ SANABRIA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO MECÁNICAS
ESCUELA DE INGENERIA DE SISTEMA E INFORMATICA
BUCARAMANGA
2011

Tabla de contenido

LISTADO DE TABLAS

LISTADO DE FIGURAS

RESUMEN

TITULO: ANÁLISIS E IMPLEMENTACIÓN DE UN MECANISMO DE TOLERANCIA A FALLAS Y RESTAURACIÓN PARA UNA INFRAESTRUCTURA DE CÁLCULO DISTRIBUIDO EN REDES UNIVERSITARIAS[1]

AUTOR[2]: Rosemberg José Uribe Espinosa.

PALABRAS CLAVE: Clúster, Computemode[3], Tolerancia a fallas, Computación de alto rendimiento, Puntos de restauración (Checkpoints), Recursos ociosos, núcleo (Kernel).

DESCRIPCIÓN:

Poder contar con un nivel de tolerancia a fallas que se presentan en un momento dado dentro de estructuras clúster es indispensable para que el tiempo de cómputo utilizado junto con los cálculos realizados no se pierdan.

En la creación de un clúster se usan herramientas que conforman lo que se conoce como un middleware que permite la calendarización y administración de recursos. OAR es la herramienta que administra los recursos que hacen parte del clúster facilitando de cierta manera su creación y manejo, mientras que Computemode es una herramienta que permite crear un clúster ligero a partir de recursos ociosos administrándolos con OAR. Ninguna de estas herramientas cuenta con un sistema que le permita responder a fallas de infraestructura eléctrica, nodos, equipos de redes, entre otras, lo que crea inconformismo en los usuarios que ven a esta pérdida de tiempo de cómputo como un atraso en sus proyectos.

Dentro de la tolerancia a fallas, la creación de *checkpoints* (puntos de chequeo) es una solución que permite presentar una respuesta a más de una falla y a la vez crea un sentimiento de seguridad en el usuario de la estructura de computación de alto rendimiento.

Al contar con un respaldo de la información que se va obteniendo, se incrementaría el uso de estas plataformas, al aumentar la confianza en el uso de estas estructuras, especialmente en un Clúster basado en Computemode.

Este proyecto busca realizar una investigación sobre los tipos de fallas que se pueden presentar categorizarlas y buscar una posible solución, además de las diferentes opciones de creación de checkpoints, su aplicabilidad, funcionamiento y compatibilidad para escoger la más apropiada e integrarla a Computemode y así dotarlo de un sistema de tolerancias a fallas.

[1] Trabajo de grado. Modalidad: Trabajo de investigación.

[2] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática.
Director: Ing. Juan Carlos Escobar Ramírez, Codirector: PhD Carlos Jaime Barrios Hernández.

[3] http://computemode.imag.fr

SUMMARY

TITTLE: ANALYSIS AND IMPLEMENTATION OF A FAULT TOLERANCE MECHANISM AND RESTORATION FOR DISTRIBUTED COMPUTATON INFRASTRUCTURE AT UNIVERISTY NETWORKS.[4]

AUTHOR[5]: Rosemberg José Uribe Espinosa.

KEYWORDS: Cluster, computemode[6]., fault tolerance, high performance computing, checkpoints, idle resources, kernel

ABSTRACT

Able to have a fault tolerance level that occur in a given time in those structures is essential for the computing time used along the done calculation do not get lost

In a cluster creation are used tools known as a middleware that allows scheduling and resources management. OAR is a tool that administrates the resources that are part of the cluster, facilitating its creation and management, while Computemode is a tool that allows a light cluster creation from idle resources, managing it with OAR. None of these tools count with a system against electric infrastructure failure, nodes, network equipment, among others, creating nonconformity among users that see this computing time lost like a delay in his projects.

Inside the fault tolerance, the checkpoints creation is a solution that allows present a response to more than a failure and at the same time it create a safety sense at the user about the high performance computing structure.

Having a backup of the information to be obtained, would increase the use of this platforms, increasing the structure use confidence, especially in a Computemode based cluster .

This project aims to conduct an investigation about the types of failures that may occur, categorize and find a possible solution, besides the different checkpoint creation options, its applicability, operation and compatibility to choose the most appropriate and integrate it to Computemode and give it a fault tolerance system

[4] Thesis Investigation

[5] Phisycs adn Mechanical engineering faculty, Systems Engineering and Informatic School. Manager: Ing. Juan Carlos Escobar Ramírez, Co-manager: PhD Carlos Jaime Barrios Hernández.

[6] http://computemode.imag.fr

INTRODUCCIÓN

Las actividades científicas y académicas de una universidad requieran hoy en día del uso de cómputo científico para el desarrollo de la enseñanza y el avance de investigaciones, esto implica el uso de equipos que cuenten con procesamiento en paralelo lo que les otorga un gran poder de procesamiento lo frustrante es que los recursos que cuentan con estas propiedades son de un alto costo por lo cual no son adquiridos por la universidad, estos recursos de procesamiento necesarios pueden obtenerse mediante el uso de los recursos presentes en la universidad dentro de las salas de cómputo e investigación aplicando mecanismos de coordinación tales como lo son los clústers y grids.

Los clústers y grids, permiten el uso de recursos menos potentes y de un menor costo para construir infraestructuras de cómputo de alto rendimiento (HPC), que permiten realizar investigaciones que requieran gran capacidad de cómputo y almacenamiento. en la actualidad la universidad cuenta con una infraestructura HPC, dotada de clústers de dos tipos; clústers dedicados, disponibles en cualquier momento y que utilizan un manejador de recursos llamado OAR , y clústers ligeros para la utilización de recursos ociosos que se manejan utilizando herramientas como Computemode y OAR, que se encuentran disponibles en horarios fuera de su uso normal (Horarios nocturnos, festivos , sábados y domingos), el creciente interés de la comunidad investigadora dentro y fuera de la universidad en el uso de esta herramienta hace necesario mejorar su estado tanto de servicio como de seguridad ya que son sensibles a presentar fallas tales como, la pérdida de un nodo, daños en la estructura de red y en ocasiones la suspensión de servicio eléctrico. Todas estas se presentan por diferentes factores, lo preocupante es que los trabajos que se desarrollan en el momento de presentarse la falla, deben ser reiniciados y el tiempo de cómputo ya utilizado se pierde.

Este proyecto busca analizar las fallas que se presentan en este tipo de infraestructura y dotarla de un método de tolerancia a fallas mediante la implementación de checkpoints que permita a los usuarios tener confianza en el uso de esta herramienta, para la realización de investigaciones actuales y futuras.

1. ESPECIFICACIONES DEL PROYECTO

TITULO

ANÁLISIS E IMPLEMENTACIÓN DE UN MECANISMO DE TOLERANCIA A FALLAS Y RESTAURACIÓN PARA UNA INFRAESTRUCTURA DE CÁLCULO DISTRIBUIDO EN REDES UNIVERSITARIAS

DIRECTOR

ING. JUAN CARLOS ESCOBAR RAMIREZ

Universidad Industrial de Santander, Bucaramanga Colombia.

CODIRECTOR

Ph.D CARLOS JAIME BARRIOS HERNANDEZ

Universidad Industrial de Santander, Bucaramanga Colombia.

AUTOR

ROSEMBERG JOSÉ URIBE ESPINOSA

Estudiante de Ingeniería de Sistemas.

rosterg85@gmail.com

2 ORIENTACION SOBRE CONTENIDO DEL PROYECTO

Durante el desarrollo del documento se mostraran las diferentes etapas de desarrollo del proyecto. Investigación, Análisis del estado actual, posibilidades de solución y su análisis, desarrollo e implementación, resultados y alcances del proyecto. La información se encuentra distribuida de la siguiente manera:

RESUMEN. Breve explicación del contenido del libro resaltando las ideas mas relevantes para dar una idea del tema y desarrollo del contenido del libro.

TABLA CONTENIDO. Tabla en la que se lista todo el contenido del libro estructurado por capítulos con su respectiva numeración de página.

TABLAS DE FIGURAS Y TABLAS. Muestran las diferentes figuras y tablas del libro y su ubicación.

CAPITULO 1. Introducción: Se realiza una presentación general del proyecto, análisis del estado actual, objetivos generales, específicos y justificación.

CAPITULO 2. Especificaciones de director, codirector, Autor del proyecto contenido del proyecto y diccionario de términos relacionadas a las temáticas tratadas en el libro.

CAPITULO 3. Durante el desarrollo de este capítulo se presenta el planteamiento del proyecto, justificación del mismo y objetivos planteados.

CAPITULO 4. Marco teórico: Se presentan los diferentes conceptos relacionados con el proyecto en su parte técnica.

CAPITULO 5. Estado del Arte: En esta capitulo se muestra los avances que hasta el momento se realizan en los campos relacionado al proyecto.

CAPITULO 6. Dentro de este capítulo se presenta el desarrollo del proyecto, las investigaciones y las diferentes decisiones y opciones que se tomaron.

CAPITULO 7. En este capítulo se desarrollan las pruebas para demostrar el correcto funcionamiento de la estrategia implementada.

CAPITULO 8. Listado de las limitaciones encontradas durante el desarrollo del proyecto.

CAPITULO 9. Conclusiones del proyecto.

CAPITULO 10. Recomendaciones a futuro para la continuación y mejora del proyecto.

ANEXOS

BIBLIOGRAFIA

GLOSARIO

ARQUITECTURA ESCALABLE: Un computador, incluyendo hardware y software, se dice escalable si podemos aumentar sus recursos para soportar una mayor demanda de rendimiento y funcionalidad, y/o disminuir sus recursos para reducir costos, esta misma definición se aplica al tipo de estructuras escalables con respecto a sus componentes.

CLUSTER DISKLESS: Son aquellos que se crean por la agregación de recursos y se caracteriza por solo utilizar el poder de cómputo de nodo con esto nos referimos al procesador y la memoria RAM.

CHECKPOINT: (Punto de restauración) Palabra en inglés que define un mecanismo de recuperación que puede generar un archivo que contiene información necesaria para que un equipo pueda volver a un estado consistente.

COMPUTACION DISTRIBUIDA: Un sistema distribuido generalmente es un conjunto de equipos de cómputo interconectados donde los recursos informáticos son compartidos con todos los otros equipos en el sistema. La potencia de procesamiento, la memoria y el almacenamiento de datos, son recursos de la comunidad donde los usuarios autorizados pueden entrar y realizar ciertas tareas. Una red distribuida puede ser tan simple como una colección de equipos similares funcionando con el mismo sistema operativo, o tan complejo como una red interconectada de sistemas, formada por cualquier plataforma informática.

COMPUTEMODE: Software que permite la creación de clúster ligeros mediante la agregación de recursos de manera no intrusiva.

CONFIABILIDAD: Es la medida en la cual la confianza se puede poner justificadamente en el servicio que se obtiene del sistema (Laprie (1992))[7].

CRON: Es un administrador regular de procesos en segundo plano (*demonio*) que ejecuta procesos o guiones a intervalos regulares (por ejemplo, cada minuto, día, semana o mes). Los procesos que deben ejecutarse y la hora en la que deben hacerlo se especifican en el Archivo crontab. El nombre *cron* viene del griego *chronos* que significa "tiempo".

FIABILIDAD*: El* término "**fiabilidad**", que se utiliza en algunos casos, se refiere a la probabilidad de que un sistema funcione normalmente durante un período de tiempo dado. Esto se denomina "**continuidad del servicio**".

[7] http://www2.laas.fr/JC-Laprie/parcours-jcl.html

KERNEL: es la parte fundamental de un sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios y de llamadas al sistema.

MODULO: Modulo se refiere a un controlador de un dispositivo o servicio, que puede cargarse o descargarse cuando el usuario o algún dispositivo lo solicita.

MULTIPLEXACIÓN: Es la combinación de dos o más canales de información en un solo medio de trasmisión.

PID: El PID o process ID, es un numero que usa el sistema Unix para identificar a un proceso.

WALLTIME: Palabra en ingles que significa límite de tiempo, que es el tiempo de reserva dado para el cómputo realizado por un trabajo especifico.

3. PRESENTACIÓN DEL PROYECTO

3.2 PLANTEAMIENTO DEL PROBLEMA

Los grupos y centros de investigación de la universidad no cuentan con los equipos necesarios para la realización las investigaciones que requieren de gran poder de cómputo y al querer aplicar estrategias tales como la creación de un cluster que les permita obtener el poder de cómputo necesario para realizar sus actividades, pero estas estrategias no se puede llevar a cabo debido a la falta de equipos que se puedan disponer de forma dedicada a formar parte de un cluster.

Observando dentro de los campus universitarios encontramos una gran cantidad de equipos que no son aprovechados en su totalidad, permaneciendo ociosos durante largos periodos de tiempo (noches, fines de semana y festivos), con el fin de poder realizar un aprovechamiento de estos recursos se puede optar por el uso de una herramienta software que permita crear un clúster por agregación de recursos utilizando solamente su poder de cómputo (RAM y procesador) sin realizar cambios en el software y el hardware de los equipos, este tipo de cluster es conocido como cluster ligero y permiten el uso de los recursos de manera no intrusiva ya que al finalizar el tiempo de cómputo los equipos seleccionados para formar parte del cluster regresan a su estado normal sin ninguna alteración, lo que permite al usuario normal del equipo continuar con sus actividades. Implementar este tipo de tecnología permitiría a quienes buscan un gran poder de cómputo para realizar sus investigaciones aprovechando los recursos disponibles dentro del claustro universitario.

Los clusters ligeros y dedicados son muy sensibles a fallas de estructura como lo son daños de los equipos tanto en el hardware como en el software y fallas en la comunicación tales como daños en el cableado, tarjetas de red, perdida de routers, switches, etc. Por ser herramientas de investigación cualquier falla que se produzca retrasaría el desarrollo de las investigaciones que dependen de los resultados obtenidos mediante el uso del cluster. Estas herramientas de cálculo no cuentan con un sistema que les permita reaccionar a las fallas aplicando correctivos que les permitan realizar una recuperación del estado de los procesos de cómputo que viene desarrollando y en el caso de los cluster ligeros los trabajos que manejan están restringidos al tiempo de cómputo de la plataforma (tiempo de ocio de los equipos) ya que finalizado este tiempo los equipos regresan a su estado original y los procesos que se desarrollan en ese momento se pierden. Se hace necesaria entonces la creación de un sistema de tolerancia a fallas que permitan la ejecución de procesos de larga duración, así como el restablecimiento de las tareas o procesos que no llegaron a término por diferentes circunstancias, tanto accidentales como incidentales.

3.3 JUSTIFICACIÓN DEL PROYECTO

Dentro de la universidad ya se encuentran funcionando estructuras de Cómputo de alto rendimiento desde tiempo atrás, estas estructuras están siendo usadas por diferentes alumnos y profesores para poder realizar sus actividades de investigación, que requieren gran cantidad de poder de cómputo y un tiempo de disponibilidad de recursos considerable, dentro de las estructuras creadas en la universidad, se cuenta con clústers dedicados y clústers ligeros, estos segundos cuentan con la misma herramienta de manejo de recursos que los clúster dedicados OAR y además cuentan con una herramienta que permite la agregación de recursos en su tiempo de ocio llamada Computemode, esta estructura en particular presenta limitaciones debido a su naturaleza oportunista que usa recursos ociosos, es decir, está limitada en tiempo de cómputo, cuando esta no es utilizada como herramienta por los integrantes del campus universitario.

Estos clústers no pueden ser usados para cálculos extensos por que el tiempo de uso es limitado y la herramienta actual no cuenta con un sistema que permita tomar el estado de un trabajo y almacenarlo para dar continuidad a futuro. La infraestructura CÓMPUTO DE ALTO RENDIMIENTO de la universidad en su mayoría está compuesta por *clúster disckless*, que cuentan con un buen número de recursos, pero que no son muy utilizados debido a su corto tiempo de funcionamiento.

El uso de un clúster dedicado o un clúster ligero, generaría una mayor atracción si se dotaran de un mecanismo que les permita ofrecer un servicio de respaldo a los trabajos de cómputo que este realiza, que les garantice a los usuarios que al presentarse una falla, el tiempo de cómputo ya invertido no se perderá, ya que esto se vería reflejado en sus proyectos, Pensando en esto se busca dotar a las infraestructuras HPC presentes en la universidad de un mecanismo de tolerancia a fallas que permita a los usuarios utilizar tanto los clústers dedicados como los clústers ligeros aprovechado la totalidad de la infraestructura con la confianza de tener una tolerancia a fallas comunes y que podrían retrasar el avance de sus proyectos .

3.4 OBJETIVOS

3.4.1 OBJETIVO GENERAL

Analizar e implementar un mecanismo para la creación de puntos de restauración y la gestión de fallas, en una infraestructura de cálculo distribuido basada en Computemode.

3.4.2 OBJETIVOS ESPECÍFICO

- Analizar y caracterizar la plataforma de cálculo distribuido teniendo en cuenta la infraestructura Computemode que se encuentra instalada.
- Analizar y clasificar las diferentes fallas y proponer mecanismos de solución.
- Analizar los casos en los cuales sea necesaria la creación de puntos de restauración y proponer mecanismos de generación de estos.
- Evaluar los diferentes lineamientos y mecanismos que se pueden implementar, para caracterizar y conocer el desempeño de las soluciones propuestas.
- Implementar un mecanismo de tolerancia a fallas.
- Implementar un mecanismo para la generación de puntos de restauración.
- Validar los mecanismos diseñados mediante la provocación de fallas que disparen el uso de los mismos[8].

[8] Es decir, lo que se busca es correr trabajos en una infraestructura de cálculo distribuido (clúster), bajo condiciones ideales, para luego comparar los resultados obtenidos en la misma infraestructura pero provocando fallas de fluido eléctrico y perdida de nodos por daño físico en el cableado, en dispositivos de conexión (swicht, router) o finalización del tiempo de cómputo.

4. MARCO TEORICO

4.1 CARACTERÍSTICAS DE LA COMPUTACION DISTRIBUIDA

Una infraestructura de computación distribuida debe presentar las siguientes características para ser considerada correcta en configuración y funcionamiento.

1. Para cada uno de los usuarios debe de ser similar al trabajo en el Sistema Centralizado.
2. Seguridad interna en el sistema distribuido
3. Se ejecuta en múltiples Computadoras.
4. Tiene varias copias del mismo Sistema Operativo o de diferentes Sistemas Operativos que proveen los mismos servicios.
5. Entorno de trabajo cómodo
6. Dependiente de redes (LAN, MAN, WAN, etc.)
7. Compatibilidad entre los dispositivos conectados
8. Transparencia (El uso de múltiples procesadores y el acceso remoto debe de ser transparente para el usuario).

La computación distribuida ha sido diseñada para resolver problemas demasiado grandes para cualquier supercomputadora y mainframe, mientras se mantiene la flexibilidad de trabajar en múltiples problemas más pequeños. Por lo tanto, la computación distribuida es naturalmente un entorno multiusuario; por ello, las técnicas de autorización segura son esenciales antes de permitir que los recursos informáticos sean controlados por usuarios remotos.

4.2 CLUSTER COMPUTING

Un cluster es un tipo de sistema distribuido o paralelo, que consiste en un conjunto de computadores independientes conectados entre sí como si fuesen uno solo. Integrando sus recursos de cómputo (procesador/es, memoria, etc.).

Generalmente un cluster se refiere a dos o más computadores (nodos) conectados entre sí, los nodos pueden estar en un simple gabinete o separados físicamente y conectados por una red de tipo LAN. Un cluster formado de varios equipos, parece tanto a usuarios como aplicaciones un solo sistema.

Por un bajo costo se obtienen características y beneficios, que solo se encontraban en los sistemas de alto costo provistos de memoria compartida. A continuación se muestra una grafica de la arquitectura típica de un clúster (Buyya, 1999)

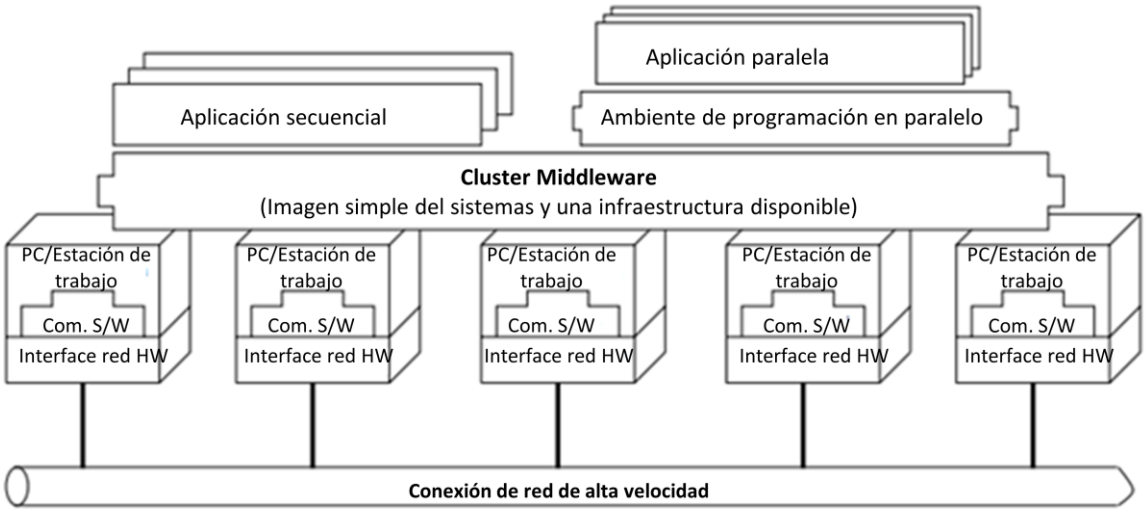


Figura 1: Arquitectura de un clúster **(Buyya, 1999)**

4.2.1 Componentes de un Clúster

A continuación se listan los componentes necesarios en un clúster:

- Varios equipos de alto rendimiento (PCs, estaciones de trabajo,o SMPs).
- Sistema operativo (en capas o basado en micro kernel).
- Redes y switches de alto rendimiento (Como Gibabit Ethernet y Myrinet).
- Tarjetas de interfaz de red (NICs).
- Protocolos de rápida comunicación y servicios.
- Clúster middleware (imagen simple del sistema “Single System Image SSI” y sistemas de disponibilidad de infraestructura).
    - ✓ Hardware (como el canal de memoria Digital (DEC), El hardware del DSM, y técnicas de SMP).
    - ✓ Kernel del sistema Operativo o Gluing Layer (tal como Solaris MC,GLU-nix).
    - ✓ Aplicaciones y subsistemas.
      - Aplicaciones (tales como la herramienta administradora del sistema y formularios electrónicos.

-Sistemas Runtime (talas como el software DSM y el sistema de archivos paralelos).

- Manejador de recursos y el software de programación (Como lo es el LSF (Load Sharing Facility) facilitador de intercambio de carga and CODINE (Computing in Distributed Networked Enviroments) computación en ambientes de redes distribuidas).

- Ambiente de Programación en paralelo y herramientas (como los compiladores PVM(Parallel Virtual Machines) maquinas virtuales de paralelización, y MPI (Message Passing Interface) interface de paso de mensajes).
- Aplicaciones
  - ✓ Secuanciales
  - ✓ Paralelos o distribuidos

Los nodos del clúster pueden trabajar colectivamente, como un recurso computacional integrado, o puede operar como un computador individual, El middelware del clúster es responsable de ofrecer la ilusión de una imagen de sistema unificado (única imagen del sistema) y de la disponibilidad de un grupo de computadores independientes pero conectados entre sí.

4.2.2 Clasificación de los clúster

Para realizar una clasificación de los clúster nos centraremos en la función específica para la que fueron construidos, y las características que debe cumplir según su función

Un clúster puede ofrecer las siguientes características a un bajo costo:

- Alto rendimiento
- Capacidad de expansión y escalabilidad
- Alta disponibilidad

La tecnología clúster permite a una organización usar su poder de cómputo actual (el hardware en materias primas y componentes de software) que adquiere en relativo bajo costo, esto otorga a la empresa expansibilidad (camino de mejora

económico que permite aumentar su poder de cálculo conservando la inversión existente sin incurrir en gastos).

El rendimiento de aplicaciones también mejora con el soporte de un ambiente de software escalable, otro beneficio del clustering es la capacidad de conmutación por error que permite a un equipo de respaldo asumir las tareas de un pc que falla dentro del clúster. Un clúster se clasifica en categorías dependiendo de varios factores como los que se explican a continuación.

- Aplicación final: en qué tipo de ambiente se desempeñara su uso
  - ✓ Alto rendimiento (HP) clúster: Este tipo de Clúster está enfocado hacia las tareas que requieren del uso de una gran capacidad computacional, gran cantidad de memoria, comúnmente requieren estas tareas el uso de los recursos por largos periodos de tiempo, lo que se busca es que el conjunto de maquinas que hacen parte del clúster funcionen como una gran máquina muy potente.
  - ✓ Alta disponibilidad (HA) clúster: Estos clúster están pensados para tener una alta disponibilidad de sus servicios mediante el uso de servidores de respaldo de información, esto los dota de flexibilidad y robustez necesarias en un entorno donde el intercambio de información, el almacenamiento de datos sensibles y disponibilidad total de servicios son necesarios en todo momento.
- Hardware que compone el nodo
  - ✓ Clúster de PCs
  - ✓ Clúster de estaciones de trabajo
  - ✓ Clúster de SMPs
- Sistema operativo del nodo
  - ✓ Linux clústeres
  - ✓ Solaris clústeres
  - ✓ NtT clústeres
  - ✓ AIX clústeres
  - ✓ Clusters de maquinas virtuales
  - ✓ HP-UX clúster
  - ✓ Microsoft Wolfpack clúster
- Configuración de los nodos:
  - ✓ Clúster Homogéneo: En este tipo de clúster los nodos tienen el mismo hardware y el mismo sistema operativo.

- ✓ Clúster Heterogéneo: En este tipo de clúster los nodos tienen diferente hardware y diferente sistema operativo.

- Tipos de nodos: como serán los nodos que lo componen dedicados solo al clúster o no.
  - ✓ Clúster dedicado: Este tipo de clúster cuenta con nodos dedicados solo a este es decir no se emplean en otra actividad diferente, Por esta razón este tipo de nodos no cuentan con un sistema operativo diferente al usado para el clúster, tampoco están dotados de monitor, pantalla o teclado y su única función es la de realizar el cómputo de datos de los diferentes trabajos enviados al clúster.
  - ✓ Clúster no dedicado: En este tipo de clúster los trabajos se ejecutan por la apropiación de espacios de tiempo de ocio de la cpu, la razón por la cual se realiza este tipo de clúster se debe al hecho de que la mayoría de los ciclos de CPU de las estaciones de trabajo no son utilizados, incluso durante las horas pico.

4.3 COMPUTEMODE

ComputeMode (CM) es una infraestructura software basada en Debian Linux que permite extender o crear un Clúster HPC agregando recursos de cómputo no usados. Esto debido a que la mayoría de los computadores en las grandes compañías y campus universitarios están ociosos durante la noche, los fines de semana, las vacaciones, por entrenamiento de personal o viajes de negocios.[9] Fue desarrollado por el equipo Mescal[10] del Laboratorio de Informática de Grenoble [11]en Francia.

Computemode basa su funcionamiento en una estructura maestro-esclavo usando el servidor central como maestro. Dispone de una interfaz de administración web para un manejo fácil, manteniendo la disponibilidad de los computadores registrados en la red de área local.

Computemode mantiene un listado de los computadores en su red local los cuales dentro de ciertos horarios no se encuentran en uso, por lo cual de común acuerdo

[9] Computemode Project http://computemode.imag.fr/mediawiki/index.php/ComputeMode_Grid_Manager
[10] *http://mescal.imag.fr*
[11] *http://liglab.imag.fr*

con los propietarios y administradores de los equipos se programa el Computemode para que tome posesión del equipo durante sus horas de ocio (noches, fines de semana, festivos, etc..) permitiendo así contar con una infraestructura de cálculo y aprovechar la capacidad de cómputo de estos equipos.

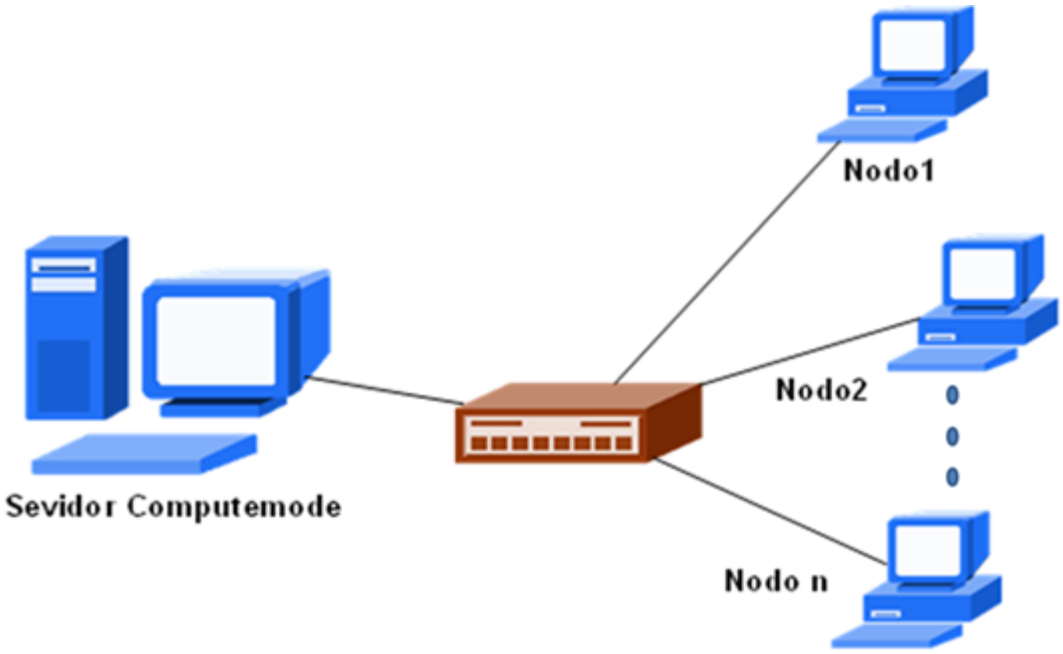


Figura 2: Estructura básica de Computemode

Computemode es una estructura como vemos en la figura compuesta por un servidor que usa un canal de comunicación por red para estar conectado con sus nodos, los cuales pueden presentarle al usuario dos tipos de estado nombrados a continuación.

- **Usermode:** Modo usuario, en este estado la maquina se encuentra en uso por el usuario común, realizando las tareas habituales, bajo su sistema operativo común, en la mayoría de casos con Windows, este estado permite al usuario realizar sus tareas sin darse cuenta de que su equipo es gestionado por Computemode como un nodo de procesamiento ComputeMode. [12]

[12] Computemode Project http://computemode.imag.fr/mediawiki/index.php/Overview

- **Computemode:** Modo de cómputo, De este estado viene el nombre de la herramienta, durante este estado los nodos hacen parte del clúster, y son usados para realizar cálculos para los diferentes usuarios. El momento en que un PC cambia de *usermode* a Computemode está definido por el administrador del clúster y se realiza mediante un reinicio automático del nodo y procede a un arranque remoto. El arranque remoto es manejado por el servidor ComputeMode con el "entorno de pre-ejecución" (PXE) de protocolo, que está disponible de forma nativa desde la BIOS de los PC desde 1999. Mientras que en el modo de la Computación, la máquina se está ejecutando bajo el sistema operativo Linux, y no tiene acceso a cualquier unidad local.

Computemode ofrece a sus usuarios diferentes herramientas para el manejo de recursos, además de dotar al sistema de la posibilidad de enviar el sistema operativo a los nodos vía red para que estos, lo carguen y realicen su trabajo, cuenta con una herramienta llamada OAR la cual es un manejador de colas cuyas funciones son:

- Reservar recursos adecuados para los cómputos.

- Planificar la ejecución de los trabajos.

Para entender aun mas como funciona Computemode debemos dividirlo en tres partes principales que lo componen El administrador de recursos, las herramientas de manejo de colas y monitoreo de recursos, los servicios de despliegue y las imágenes de sistema operativo.

4.3.1 Administrador de recursos de Computemode

Para realizar la administración de recursos, Computemode viene dotado de una interface web, dentro de esta se puede designar el nombre del nodo, el horario de funcionamiento en forma de calendario semanal y que imagen se desplegara en el nodo, estos cambios son guardados dentro de la base de datos de Computemode ya que existe una interacción continua entre la web y la base de datos.

4.3.2 Servicios de despliegue y sistemas operativos

Para realizar el cálculo los nodos necesitan contar con un sistema operativo, este sistema se le es enviado a los nodos por red usando los servicios TFTP, DHCP y NFS. Este sistema operativo está compuesto por un kernel junto con su initrd, y su sistema de archivos, téngase en cuenta que el kernel y el initrd es enviado hasta el nodo mientras que el sistema de archivos es compartido por medio de NFS.

Esta forma de funcionamiento permite agregar nodos de manera automática lo único que requiere en un sistema Windows es un cambio en el modo de arranque, este cambio consiste en ubicar la tarjeta de red como primer dispositivo, esto con el fin de que realice una petición DHCP, Esta petición es tomada por el servidor Computemode y recolecta datos del PC que realiza la petición entre estos su dirección MAC, luego el usuario usando el administrador de recursos asigna nombre y horario, a partir de ese instante las próximas peticiones realizadas por el PC se observara si se encuentra en dentro del tiempo de cómputo seleccionado por el usuario para así añadirlo o no al clúster y realizar el envió del sistema operativo.

4.3.3 Herramienta de manejo de colas y monitoreo de recursos[13]

Para el manejo de colas se cuenta con OAR (Resource Management System for High Performance Computing), este es un administrador de recursos de cómputo, el cual es flexible y simple, maneja recursos en un clúster, permite la ejecución de trabajos interactivos y de reserva. En su forma más simple un trabajo se define por un programa a ejecutar, el número de nodos a utilizar y el tiempo por el cual se necesitan los recursos. El trabajo interactivo permite que al usuario le sea asignado los recursos inmediatamente y este interactué directamente con ellos. En un trabajo de reserva, el usuario reserva los recursos para utilizarlos a determinada fecha por un espacio de tiempo definido, y es el usuario el que se encarga de conectarse al trabajo. Para monitorizar los recursos se cuenta con Monika el cual permite visualizar el estado de

[13] Tomado del proyecto ANALISIS Y DISEÑO DE UNA ESTRATEGIA DE INTERACCIÓN ENTRE RECURSOS DISTRIBUIDOS DE UNA INFRAESTRUCTURA DE CÓMPUTO EN REDES HETEROGENEAS.

los recursos mediante una interfaz web, esta se comunica constantemente con el servidor OAR.

4.3.4 Principales características y beneficios de usar Computemode

Computemode presenta por su filosofía de uso de recursos ociosos unas características y beneficios mostrados a continuación:

- **Transparencia:** Tanto para el usuario normal del equipo como para el investigador que usa el clúster, el montaje y desmontaje de la estructura es transparente, es decir no crea ninguna molestia para alguno de ellos, en especial al usuario común, ya que su equipo se usa en noches fines de semana y horarios en que el PC no está en uso por él.

- **Fácil administración:** Debido a que cuenta con una herramienta web que le permite al administrador mantenerse informado del estado de los recurso y modificar diferentes para metros como los el horario de funcionamiento, la imagen a montar entre otros.

- **Fácil implementación:** Computemode permite una fácil instalación en una infraestructura existente sin tener que realizar modificaciones a las computadoras.

4.4 FALLAS

Más allá del servicio que ofrezca un sistema informático, este sistema debe ser fiable con el fin de que los usuarios puedan utilizarlo en condiciones óptimas. Estas condiciones se cumplen siempre y cuando no se produzca una falla o si esta se produce el sistema sea capaz de responder adecuadamente (Feldgen, 2004).

Una falla se produce cuando un servicio no funciona correctamente, es decir que se genera un estado de funcionamiento anormal o que no se adecua a las especificaciones todo esto desde el punto de vista del usuario, un servicio tiene dos estados:

- servicio apropiado: cuando satisface las expectativas.
- servicio inapropiado: cuando no satisface las expectativas.

Una falla es atribuible a un error, es decir, a un mal funcionamiento en el equipo o el software usado. Pero no todos los errores conducen a una falla en el servicio, con el fin de evitar estas fallas se han elaborado varias estrategias que limitan las fallas las cuales son:

- La **prevención de errores**, que consiste en evitar errores anticipándolos.
- La **tolerancia a errores**, cuyo propósito es proporcionar un servicio de acuerdo con las especificaciones a pesar de los errores, lo que lleva a que se presenten redundancias en los procesos.
- La **eliminación de errores**, consiste en reducir la cantidad de errores por medio de acciones correctivas.
- La **predicción de errores**, se trata de anticipar errores y su posible impacto en el servicio.

Con estas estrategias se logran reducir las fallas o errores, pero cada falla tiene una causa entonces, las causas de las fallas pueden clasificarse de la siguiente manera:

Origen de la falla, la falla puede provenir de cualquiera de los aspectos que se listan a continuación:

- físicos
- diseño
- interacción

Las fallas también se pueden clasificar de acuerdo con las características siguientes:

- A su naturaleza: accidental o intencional con malicia o sin ella
- A la fase de creación en la vida del sistema: desarrollo u operación
- El lugar: interno o externo
- la persistencia: permanente o temporario

Una clasificación propuesta por Laprie[14] (1992) realiza la clasificación en dos aspectos.

- Fallas de interacción

[14] Laprie: “http://www2.laas.fr/JC-Laprie/parcours-jcl.html”

- Fallas temporales: fallas transigentes externas, fallas intermitentes

Las estrategias de control se basan en ciertas propiedades estadísticas. Creación de checkpoints (puntos de restauración o referencia) y bancos de pruebas con el fin de probar la fiabilidad requiere una comprensión de las características de las fallas reales. Por desgracia, obtener el acceso a estas características a falta de los datos provenientes de los sistemas a gran escala es difícil, ya que dichos datos son a menudo clasificados en categorías que no reflejan mucha información sobre la falla[2]

4.5 TOLERANCIA A FALLAS

En *informática*, se determina a la capacidad de un sistema de almacenamiento de acceder a información aún en caso de producirse alguna falla. Esta falla puede deberse a daños físicos o mal funcionamiento en uno o más componentes de hardware lo que produce la pérdida de información almacenada. La tolerancia a fallas requiere para su implementación que el sistema de almacenamiento guarde la misma información en más de un componente de hardware o en una máquina o dispositivo externos a modo de respaldo. De esta forma, si se produce alguna falla con una consecuente pérdida de datos, el sistema debe ser capaz de acceder a toda la información recuperando los datos faltantes desde algún respaldo disponible.

4.5.1 Tolerancia a fallos en estructuras distribuidas

Un sistema distribuido presenta fallas parciales es decir un componente se pierde pero el sistema no colapsa, en los sistemas no distribuidos las fallas son totales donde un componente afecta el funcionamiento de los demás componentes que hacen parte del sistema. Un objetivo de los sistemas distribuidos es poder recuperarse de las fallas parciales sin afectar el rendimiento global.

4.5.1.1 Conceptos básicos de tolerancia a fallas

Un sistema tolerante a fallas debe cumplir con los siguientes conceptos fundamentales de la tolerancia a fallas:

Disponibilidad: Propiedad del sistema que le permite estar listo para ser usado en cualquier momento

Confiabilidad: Propiedad del sistema de ejecutar servicios continuamente sin presentar fallas.

Seguridad: Si el sistema falla temporalmente se espera que nada catastrófico suceda.

Mantenibilidad: Que tan fácil se repara o recupera un sistema cuando falla.

Recuerde, un sistema altamente disponible no necesariamente es altamente confiable.

Un sistema distribuido puede ser diseñado de forma tal que sea tolerante a fallas, en un sistema distribuido tolerante a fallas, cuando parte del sistema falla, todo el sistema sigue funcionando, esta es una característica importante de los sistemas distribuidos a veces incluso más importante que todas las consideraciones de rendimiento y escalabilidad, No solo son las fallas físicas que se presentan en dicho sistema que se pueden prevenir mediante la realización de un mantenimiento de rutina en el tiempo de inactividad de la estructura por parte de los administradores. El usuario también puede tomar parte cuando este se encuentra sin conexión para realizar tareas de mantenimiento, La implementación de un sistema de tolerancia a fallas puede aumentar la fiabilidad y la disponibilidad de una infraestructura (Charles Fan, 2001).

¿Cómo se puede construir un sistema distribuido tolerante a fallas? No es fácil, se han desarrollado estudios tanto en la teoría de modelado de los sistemas distribuidos, como en la práctica y aplicación de dichos sistemas dando como resultado publicaciones de estudios realizados por expertos en búsqueda de una mejor comprensión de las estructuras distribuidas para poder realizar mecanismos de tolerancia a fallas.

Para realizar un sistema distribuido tolerante a fallas se necesita conocer los diferentes tipos de fallas, a continuación se lista una clasificación de las fallas más relevantes en un sistema distribuido.

Falla permanente de un procesador: Se presenta cuando un procesador falla por interrupción y permanece en este estado.

Falla temporal en el procesador: El procesador falla por interrupción durante un periodo de tiempo.

Falla permanente de conexión: La comunicación entre dos procesadores falla por omisión de todos los mensajes que se transmitían por el enlace de conexión.

Falla temporal de conexión: La comunicación entre dos procesadores falla por omisión de un subgrupo de mensajes que se transmitían por el enlace de conexión.

Falla Bizantina o arbitraria: Se presenta cuando un procesador falla por exhibir un comportamiento arbitrario.

En un sistema asíncrono, cada operación puede realizarse a una velocidad arbitraria, esto hace que una detección fiable de fallas sea imposible ya que en ningún momento se puede saber si un procesador está funcionando muy lento debido a que no se tiene un punto de comparación para la velocidad de funcionamiento del procesador, entonces, si se produce una falla, esta no es detectada hasta que no ha pasado un tiempo considerable o por que la aplicación informa que no pudo finalizar mas no por que el sistema la detecto.

La detección de fallas es el componente central del control de fallas, contar con un buen sistema de detección de fallas le permite al sistema reaccionar al evento que produce la falla, la detección de fallas es un componente importante y esencial para la comunicación del grupo de nodos que componen la estructura, La comunicación en estas estructuras se produce por grupos lo que hace más difícil su control para lograr fiabilidad en la comunicación. La fiabilidad debe ser garantizada aun si se presentan alguno de los siguientes escenarios: un nuevo nodo se une al grupo; un nodo existente de manera voluntaria dejo el grupo; un nodo existente deja el grupo debido a una falla, Para evitar esto los nodos deben mantener un acuerdo sobre la permanencia en el grupo , el no contar con este mecanismo se conoce como problema de pertenencia de grupo (GMP), El lograr un acuerdo de participación de grupo es un requisito previo fundamental para lograr la comunicación, hay tres aspectos que hacen parte de una comunicación fiable:

La entrega confiable: Si un receptor está habilitado para recibir mensajes, con el tiempo debe recibir todos los mensajes que se le enviaron.

Entrega Atómica: Cuando un mensaje es enviado a un grupo, este deberá llegarle correctamente a todos los miembros del grupo o a ninguno de ellos.

Pedido de mensajes coherentes: Cuando un grupo recibe una serie de mensajes. Los nodos en el grupo deben recibirlos en el mismo orden.

Con lo visto anteriormente queda claro que la tolerancia a fallas busca mantener control sobre dos aspectos, el funcionamiento correcto del procesamiento en cada nodo y la comunicación entre los diferentes componentes de la estructura.

Dentro de los mecanismos de tolerancia a fallas encontramos la migración de procesos y la realización de checkpoints estos métodos permiten realizar una rápida recuperación del sistema lo que se traduce en una alta manteniblidad. A continuación se describen con mas detalle estos métodos.

4.6 MIGRACIÓN

La migración de procesos es el acto de transferencia de un proceso entre dos máquinas (La fuente y el nodo de destino) durante su ejecución. Algunas arquitecturas también definen un host o nodo inicial, que es el nodo donde se ejecuta el proceso lógico (Milojicic, Dougiles, Paindaveine, Wheeler, & Zhou, 2000)

. Un alto nivel de la migración de proceso se muestra en la siguiente figura.

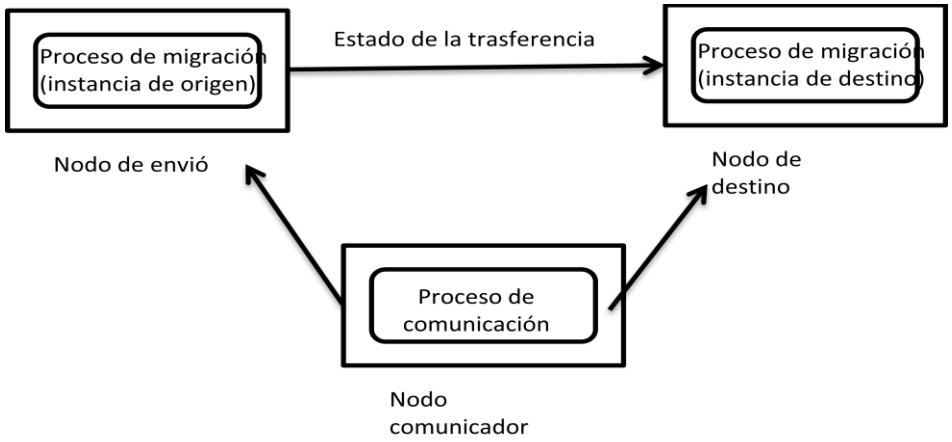


Figura 3: Proceso de migración Imagen tomada y traducida del artículo Process Migration incluido en la referencia del capitulo

El proceso de transferencia de estado incluye el espacio de dirección, punto de ejecución (el contenido del registro), la comunicación del Estado (por ejemplo, los archivos abiertos y canales de mensajes) y otro sistema operativo dependiendo del estado, la migración de tareas representa la transferencia de una tarea entre dos equipos durante la ejecución de sus hilos. Existen dos instancias del proceso de migración la instancia de origen que es el proceso original y la instancia de destino que es el nuevo proceso que se creó en el nodo de destino Después de la migración, la instancia de destino se convierte en un proceso de migración. En los sistemas con un nodo de origen, un proceso que se ejecuta en otros equipos puede ser llamado proceso remoto (desde la perspectiva del nodo de inicio) o un proceso de extranjeros (desde la perspectiva del nodo de alojamiento).

4.7 CHECKPOINTS (PUNTOS DE RESTAURACIÓN)

El desarrollo de checkpoints es un importante esfuerzo hacia la tolerancia a fallas, este provee al programador la capacidad de tomar una imagen instantánea del estado de la aplicación, periódicamente o por comandos. En el momento en que ocurra una falla en el sistema, estos datos de checkpoint pueden ser usados para restaurar la aplicación y que la aplicación avance a partir de este punto. La mayoría de los checkpoints basados en la tolerancia a fallas, exige al desarrollador de la aplicación el escribir código adicional para que se realice la creación del checkpoint y para realizar el reinicio por medio de su uso, para realizar tareas de escritura y lectura de archivos en disco y volver a reinicializar las estructuras de datos. Por otra parte, suele suponerse que el reinicio se realiza con el mismo número de procesadores que se usaron para crear el punto de restauración. Con este supuesto, el usuario tiene que esperar un tiempo de respuesta para la restauración de los nodos, o contar con la disponibilidad de un conjunto de nodos adicionales que se puedan usar para sustituir los perdidos. En otras palabras la tolerancia tradicional basada en la tolerancia a fallos implica un importante aumento en la complejidad de la programación Y/o recursos de hardware. En los métodos basados en checkpoint, El estado de la computación es salvado como un checkpoint periódicamente en un almacenamiento estable, de acuerdo con el tipo de coordinación entre los diferentes procesos en el momento de crear el checkpoint, los métodos de creación de checkpoint se pueden clasificar en tres categorías (Zheng, Huang, & V. Kal, 2006):

- creación de checkpoint no coordinada
- creación de checkpoint coordinada
- creación de checkpoint por comunicación inducida.

Las implementaciones de checkpoint/restart (creación de checkpoint y reinicio de procesos mediante el uso de checkpoint) pueden clasificarse en dos tipos, las de nivel de sistema y las de nivel de usuario.

- Las aplicaciones a nivel de sistema requieren que sus librerías se compilen como parte del kernel o que carguen como módulos del kernel, ya que necesitan acceder a datos del kernel del sistema.

- Las aplicaciones a nivel de usuario requieren modificaciones mínimas del kernel del sistema operativo. Pero la mayoría requieren ya sea pre-procesamiento del código fuente, o vinculación del código objeto a la rutina de la librería. Además, ya que los programas de nivel de usuario no pueden acceder al kernel del sistema, y en la práctica por lo general tienen menos capacidades que las implementaciones a nivel de sistema como los son el

descriptor de archivos, sockets, etc. Estas implementaciones no pueden ser guardadas debidamente en los checkpoints (Cao, Li, & Guo, 2005).

4.7.1 Creación de Checkpoint no Coordinada

En este tipo de creación de checkpoint, cada proceso de forma independiente guarda su estado. Durante el reinicio, estos procesos buscan entre el conjunto checkpoints guardados un estado coherente para que la ejecución pueda reanudarse. La ventaja de este esquema es que un checkpoint tiene lugar cuando este sea más conveniente. Por eficiencia un proceso puede llevar a cabo un checkpoint cuando el estado del proceso es pequeño (Wang, 1993). Sin embargo la falta de coordinación entre los checkpoint es susceptible a la rollback propagation, el efecto *domino*, que podría provocar que el sistema regresase al principio de la computación resultando la perdida de una gran cantidad de trabajo útil. El rollback propagation también hace que sea necesario almacenar múltiples checkpoints en cada procesador, que puede conducir a una sobre carga de almacenamiento. Este costo hace inadecuado este método de checkpoint para un clúster ligero.

4.7.2 Creación de Checkpoint Coordinada

Este método requiere procesos que coordinen los checkpoints a fin de formar un estado global consistente, coordinando los puntos de restauración con lo que se logra la simplificación de la recuperación de fallas porque estos no sufren la rollback propagation, además minimiza el costo de almacenamiento ya que sólo realiza un checkpoint cuando este se necesita.

Para poder coordinar los checkpoints se usa un proceso especial, que maneja la forma en que se salvan los procesos así como los mensajes que llegan hasta que estos finalizan.

4.7.3 Creación de Checkpoint por Comunicación Inducida

Permite que los procesos tomen checkpoints de manera individual para así evitar el efecto dómino. Obligando los procesadores a tomar checkpoints adicionales con base en la información relacionada al protocolo de instancias de la aplicación por

medio de los mensajes que recibe de otros procesadores (Briatico, Ciuffoletti, & Simoncini.).Sin embargo, el checkpoint debe tomarse antes que la aplicación pueda procesar el contenido del mensaje, posiblemente llevando a una alta latencia y gastos generales.

4.8 DISKLESS CHECKPOINTING

Diskless checkpointing[15] es una técnica usada para tomar una imagen del estado de un programa en un sistema distribuido sin depender de almacenamiento estable. Sustituye al almacenamiento estable con el uso de la memoria y la redundancia del procesador. Esta técnica genera una alta sobrecarga de memoria en el momento de almacenar los checkpoints. Los autores del articulo Faster checkpointing (Plank & Li, 1994), presentan una forma de realizar puntos de comprobación rápida, progresiva mediante el uso de N + 1 pares de procesadores con el fin de aliviar este problema. El algoritmo elimina el almacenamiento estable y la escritura en disco. Todos los procesadores en forma cooperativa mantienen checkpoints locales de un estado global consistente. Un procesador de checkpoint y un procesador de copia de seguridad están reservados para el almacenamiento, a estos se les llama puesto de control de la paridad. Este se calcula mediante la aplicación de operación XOR en todos los puestos de control local de cada procesador. Cuando un procesador falla, los procesadores sobrevivientes pueden recuperarse de su checkpoint local anterior en la memoria y el procesador calcula su checkpoint de todos los otros puestos de control, y desde el puesto de control de paridad, cada procesador se comunica con el procesador de la paridad lo que genera sobre carga en la red, entre otras razones. El protocolo también requiere de dos procesadores de almacenamiento adicionales para formar la paridad, así como los procesadores de espera para sustituir a los procesadores en los que falla la aplicación.

[15] Disckless checkpointing: Recuperación sin disco

4.9 BERKLEY LAB CHECKPOINT / RESTART (BLCR)

Berkeley LabCheckpoint / Restart (BLCR), es parte de los sistemas escalables Software Suite, elaborado por el Grupo de Tecnologías de Futuro en el Lawrence Berkeley National Lab con financiación SciDAC del Departamento de Energía de Estados Unidos. Es un código abierto que permite a los programas que se ejecutan en Linux realizar checkpoints que pueden ser usados durante el reinicio de la máquina e incluso en otra máquina diferente. BLCR se implementa como un modulo de Kernel cargable de licencia GPL y una pequeña biblioteca con licencia LGPL.

Dentro de sus potencialidades se encuentra el uso de los checkpoint a nivel de sistema, que ofrece beneficios en un entorno HPC que los checkpoint de nivel de aplicación no pueden. El carácter preventivo de los checkpoint a nivel de sistema permite más libertad en la programación de procesos por lotes lo que aumenta su usabilidad y reduce el tiempo de espera promedio de colas. Además, un checkpoint preventivo se puede utilizar para tolerancia a fallos en respuesta a "los precursores de fracaso" y no para los checkpoints periódicos. Es esta programación y los beneficios de tolerancia a fallas, combinada con la prevalencia de las agrupaciones Linux en HPC que motivaron la decisión de realizar un checkpoint a nivel de sistema para reiniciar la aplicación en los clústers Linux.

El proyecto a avanzado tanto que ya se puede encontrar que aplicaciones como MPICH, LAM/MPI, OPENMPI, entre otras, que cuentan con integración a BLCR durante su instalación, es decir vienen dotadas de la posibilidad de intégralas a BLCR en el momento de configurar su instalación, estas se integran a BLCR de forma tal que incluyen dentro de sus comandos de manejo de trabajos comandos de realización de checkpoints los cuales trabajan utilizando el sistema BLCR.

BLCR deja de funcionar en el kernel de Linux como un parche o un módulo cargables y en el espacio de usuario como un contenedor alrededor de la biblioteca estándar de C. las razones son las siguientes, BLCR eliminó la opción de un parche para el kernel debido a los cambios frecuentes a las fuentes del kernel, los contenedores alrededor de la biblioteca C se han descartado debido a preocupaciones sobre el costo de rendimiento de la intervención asociada a la virtualización. La decisión de poner en práctica BLCR al nivel de kernel permite el acceso a los datos que, desde el espacio de usuario, o son imposibles de modificar (por ejemplo, pid) o de muy difícil acceso (por ejemplo, nombre de archivo de un proceso abierto) (H. Hargrove & C. Duell, 2006).

4.10 DMTCP: Distributed MultiThreaded CheckPointing

DMTCP (checkpointing de multiprocesos distribuidos) es una herramienta para la creación de puntos de restauración, del estado de un grupo de programas arbitrario, repartidos en varias maquinas y conectadas por sockets. Se ejecuta y se utiliza directamente, como un ejecutable binario en el usuario sin necesidad de modificar ya sea el binario del usuario o el sistema operativo.

Dentro de las aplicaciones soportadas por DMTCP encontramos OpenMPI, MATLAB, Python, Perl, y muchos lenguajes de programación y lenguajes de scripts de Shell, DMTCP esta a nivel de usuario, no requiere privilegios de sistema para operar. Esto permite a DMTCP incluirse con la aplicación y no mediante la apertura de aplicaciones nuevas para checkpointing, DMTCP permite la migración de procesos, lo que permitiría realizar la fase intensiva de CPU y luego migrar el cálculo a una sola computadora portátil, para un análisis interactivo en este, otra parte que gusta mucho a los usuarios, es que permite la fácil depuración de trabajos de larga duración, ya que cuando se descubre un error durante la ejecución de un trabajo, el programador en repetidas ocasiones puede reiniciar desde un punto de control tomado justo antes de que el error ocurriera y examinar el fallo en un depurador. Esto reduce el ciclo de error-recompile para estos casos.

DMTCP como aplicación de nivel de usuario tiene la ventaja de que no debe ser recompilado nunca para que funcione con el kernel reciente, a diferencia de las aplicaciones de nivel de sistema las cuales deben ser compiladas usando los códigos fuente del kernel lo cual limita su uso a la versión soportada por la instalación de la aplicación de checkpoint usada (J., C., & Cooperman, 2009).

5. ESTADO DEL ARTE

En diferentes universidades del mundo, el uso de plataformas escalables para la realización de cómputo de alto rendimiento ha tenido un amplio desarrollo, y mejoras constantes. La Universidad Industrial de Santander - UIS también se ha unido a esta iniciativa y desde hace algunos años se han desarrollado proyectos con miras a proveer soporte a las necesidades de cómputo presentadas por los diferentes grupos de investigación que hacen parte de la universidad, algunas de estas soluciones implicaron la construcción de plataformas dedicadas, otras soluciones llevaron al desarrollo de clusters ligeros, tratado dentro de los proyectos citados en [16 y 17] desarrollados por alumnos de la universidad.

Los clústers ligeros que se encuentran hoy en día funcionando dentro de la universidad se desarrollaron usando la herramienta Computemode, obteniendo una estructura de de cómputo de alto rendimiento no invasiva, una de las falencias de esta estructura es la falta de un sistema de tolerancia a fallas, que permita recuperar el estado en el que se encontraba antes de que se presentara la misma, ya sea por falta de suministro eléctrico, fallo en los componentes de la estructura o simplemente que el tiempo de cómputo del nodo llego a término y debe volver a su estado original, con el fin de solucionar este inconveniente se inicia este proyecto, para analizar las fallas que se presentan en este tipo de infraestructura categorizándolas obteniendo una mejor comprensión de estas para así seleccionar las fallas que se analizaran más a fondo procediendo a analizar las diferentes estrategias de tolerancia a estas, Luego de realizado este análisis seleccionar la alternativa más adecuada para ser aplicada en la infraestructura existente en la universidad, la cual cuenta con dos clúster diskless heterogéneos compuestos por una cantidad considerable de recursos.

En el desarrollo del proyecto[11] se logro vincular recursos de clusters dedicados y clusters creados con Computemode los cuales se encuentran en redes privadas, para realizar cálculos, distribuyendo los diferentes trabajos en recursos de diversas redes, tengamos en cuenta que estas estructuras son heterogéneas y que este sistema se

[16] Análisis e implementación de una infraestructura de cálculo distribuido en la red universitaria Desarrollado por el ing. Cristian Camilo Ruiz Sanabria. (2009)

[17] Análisis y diseño de una estrategia de interacción entre recursos distribuidos de una infraestructura de Cómputo de redes heterogéneas desarrollado por los alumnos Mireya Carolina Mantilla Serrano y Sergio Andres Orostegui Prada. (2011)

encuentra aplicado hoy en día en la infraestructura de cómputo de alto rendimiento de la universidad.

6. DESARROLLO DEL PROYECTO

Durante el desarrollo del proyecto se presentaron dos etapas, una etapa de investigación durante la cual se analizaron las fallas, este análisis se realizo con el fin de identificar cuáles de estas fallas se iban a abordar en profundidad para buscarles algún mecanismo de tolerancia, luego del análisis de fallas se prosigue con la investigación de las diferentes opciones para solucionar las fallas seleccionadas durante el análisis anterior, esta investigación nos lleva a conocer las diferentes opciones y a elegir la más apropiada según el análisis de cada una de las opciones para ser acoplada a la estructura de cómputo de la universidad. Luego se ingresa a la segunda fase, la fase de implementación de la opción o estrategia elegida.

El proyecto se desarrolla dentro de una plataforma ligera construida usando Computemode pero la solución que se desea implementar se extiende también a las plataformas o clústers dedicados, Cada clúster cuenta a su vez con sus respectivos nodos y características, las infraestructuras de clúster ligeros de la universidad cuentan con las siguientes especificaciones.

Nombre del Cluster	Numero de nodos	Ubicación
CMcentic	81nodos	CENTIC
CMjavs	24nodos	EISI – Laboratorio Villabona

Tabla 1: Descripción de los clústers usados

Estos clúster son usados en horas fuera de clase (De 8 pm a 6 am), vacaciones y fines de semana. Cada servidor cuenta con servicios de administración y configuración de los nodos de cómputo además de los servicios necesarios para despliegue y monitoreo. Las especificaciones de los equipos son los siguientes:

CLUSTER CMCENTIC

Referencia Nodos	Arquitectura	RAM	Tarjeta de video	Cores	GPUs	Interfaz de Red	Cantidad de nodos
DELL OPTIPLEX GX620	3.8 GHz intel Pentium 4	2.0 GB DDR3	ATI RADEON X600	2	0	Gigabit Ethernet	81

Tabla 2: Descripción nodos clúster CMCENTIC

CLUSTER CMJAVS

Referencia Nodos	Arquitectura	RAM	Tarjeta de video	Cores	GPUs	Interfaz de Red	Cantidad de nodos
DELL OPTIPLEX 740	2.4 GHz AMD Athlon 64x24600	2.0 GB DDR3	ATI REDEON XT2400	2	0	2 x Broadcom Gigabit Ethernet	24

Tabla 3: Descripción nodos clúster CMJAVS

Luego de conocer las estructuras dentro de las cuales se realizara el proyecto procedemos a iniciar con la realización del mismo empezando por la fase de investigación.

6.1 CATEGORIZACIÓN DE ERRORRES

Durante esta sección del proyecto se analizan los errores presentes en la estructuras HPC y en el manejador de colas OAR, con el fin de obtener una mayor comprensión y abstracción de los diferentes tipos de errores, parte del estudio realizado quedara depositado en tablas para su fácil interpretación.

6.1.1 Errores frecuentes en estructuras HPC

Teniendo en cuenta que se busca extender la infraestructura de cálculo de alto rendimiento de la universidad y conectarla a otras existentes con miras a lograr una Grid, se empezó por analizar las fallas que se pueden presentar dentro de una infraestructura Grid para tenerlas en cuenta en el momento de analizar las diferentes alternativas de tolerancia a fallas, obteniéndose la siguiente categorización

- Fallas de red: Perdida de paquetes, corrupción de paquetes.
- Fallas de tiempo: Primeros fallas y fallas de última hora.
- Fallas de respuesta: Fallas de valor, fallas en el estado de transición.
- Fallas por omisión.
- Fallas físicas: Fallas de CPU, memorias, almacenaje, etc.

- Fallas de ciclo de vida: fallas por control de versiones.
- Fallas de interacción[18].

Teniendo esta categorización se realizo una categorización similar dentro de la infraestructura (Computemode) usada en la universidad teniendo como resultado la siguiente lista:

- Errores de Red: Perdida de paquetes y corrupción de paquetes.
- Fallas de tiempo: Early faults y Late faults.

  ✓ Early faults: mala codificación, errores de creación de variables, errores de sintaxis en el envío de trabajos.
  ✓ Late faults: son todas las fallas de rendimiento, como la falta de memoria, sobre carga del procesador, desborde de variables , etc.
- Fallas de respuesta: fallas de valor y fallas de transición de estado.
- Fallas por omisión: se presenta cuando un equipo no está disponible, debido a la naturaleza dinámica de este tipo de plataformas.
- Fallas físicas: Daños en el hardware del nodo y las fallas de estructura de red (cableado, swicht, tarjetas de red, etc.).

A continuación se organizaron las fallas en la tabla siguiente, y se realizo una breve explicación para su fácil entendimiento, junto con esto se propuso una alternativa de solución a cada una de las fallas incluidas en la tabla.

Error	Breve Explicación	Solución
Errores de red	Perdida de paquetes y corrupción de paquetes.	La solución la proporcionan los sistemas MPI y OAR
Fallas de tiempo	Early Faults: mala codificación, errores de creación de variables, errores de sintaxis en el envío de trabajos. Later Faults: son todas las fallas	Eary Faults: La solución consiste en que el cliente detenga el trabajo, realice las correcciones de código, vuelva a compilar y lance el trabajo. Later Faults: La solución esta en

[18] Clasificación tomada del articulo Fault Tolerance within a grid enviroment. Realizado por Paul Townend y Jie Xu , Del departamento de ciencias de la computación , Universidad de Durham Reino Unido.

	de rendimiento , como la falta de memoria, sobre carga del procesador, desborde de variables , etc	detener el trabajo, revisar el tipo de variables y como se están usando, recompilar y lanzar de nuevo el trabajo
Fallas de respuesta	Valoración de Fallas y cambio de estado por fallas	El poder valorar o analizar las fallas esta incluido en OAR, SQL, y demás sistemas. Pero los almacena en logs de no fácil acceso sin conocimiento previo, el cambio de estado se presenta en algunas ocasiones oportunamente, otras depende del usuario ocasionar este cambio de estado
Fallas por omisión	Se presenta cuando un equipo no está disponible debido a la naturaleza dinámica de computemode.	Creación de checkpoints para reinicio de trabajo cuando haya disponibilidad de recursos.
Fallas físicas	Daños en el hardware del nodo, fallas de estructura de red (cableado, swicht, tarjetas de red, etc.). Pueden tener solución pero en algunos casos se debe sustituir el elemento dañado.	Creación de informes de errores que permitan localizar el daño para que el administrador pueda repararlo, los checkpoints permiten recuperar estos trabajos si el daño del hardware lo permite.

Tabla 3: *Fallas en estructuras HPC*

En la tabla anterior realizó una síntesis de los diferentes tipos de errores presentes en las estructuras HPC, y puede ser utilizada como una valiosa herramienta para los diferentes administradores de este tipo de sistemas y para los desarrolladores como un punto de referencia al momento de realizar una implementación.

6.1.2 Categorización de errores y eventos presentes en OAR

Luego de analizar las diferentes fallas que se presentan en las estructuras, se continuó con el análisis de la herramienta encargada del manejo de colas OAR ya que esta presenta gran cantidad de posibles errores por lo cual se realizo un análisis

de estos para saber cuáles debían ser solucionados por el administrador y cuáles de estos podían ser solucionados por el usuario.

A continuación se listan y se traducen los errores o eventos informados por el oar, junto a una breve explicación del error.

1. "PING_CHECKER_NODE_SUSPECTED”: El sistema a detectado a través del modulo “Finaud” que un nodo no está respondiendo.
2. "PROLOGUE_ERROR": Un error ocurrido durante la ejecución del prologo del trabajo (exit code !=0)
3. "EPILOGUE_ERROR": Un error ocurrido durante la ejecución del epilogo del trabajo (exit code !=0)
4. "CANNOT_CREATE_TMP_DIRECTORY": OAR no puede crear el directorio donde todos los archivos de información serán almacenados.
5. "CAN_NOT_WRITE_NODE_FILE" : El sistema no fue capas de escribir el archivo que tenía que contener la lista de nodos en el primer nodo.(/tmp/OAR_job_id)
6. "CAN_NOT_WRITE_PID_FILE": El sistema no fue capaz de escribir el archivo que tenía que contener el PID del proceso oarexec en el primer nodo (/tmp/pid_of_oarexec_for_job_id).
7. "USER_SHELL" : El sistema no puede obtener información sobre la shell del usuario en el primer nodo.
8. "EXIT_VALUE_OAREXEC" : El proceso oarexec finalizó con un codigo de salida desconocido.
9. "SEND_KILL_JOB" : Indica que el oar a transmitido una señal de (kill) matar al oarexec del trabajo especificado.
10. "LEON_KILL_BIPBIP_TIMEOUT" : El modulo Leon detecto que algun error ocurrio durante la matanza de un trabajo y para matar el proceso BipBip locales.
11. "EXTERMINATE_JOB" : El modulo Leon detecto que algun error ocurrio durante la matanza de un trabajo entonces se elimino la base de datos y termino el trabajo artificialmente.
12. "WORKING_DIRECTORY" : El directorio desde el que ejecuto el trabajo no existe en el nodo asignado por el sistema.
13. ''OUTPUT_FILES" : OAR no puede escribir los archivos de resultados (stdout y stderr) en el directorio de trabajo.
14. "CANNOT_NOTIFY_OARSUB" : OAR no se puede notificar el proceso oarsub para un trabajo interactivo (tal vez el usuario ha matado a este proceso).
15. "WALLTIME" : El trabajo alcanzo su tiempo máximo de cómputo.
16. "SCHEDULER_REDUCE_NB_NODES_FOR_RESERVATION" : Esto significa que no hay nodos suficientes para la reserva, así que el programa le asigna menos nodos que los reservados (esto se presenta cuando los nodos están

suspendidos o Ausentes).

17. "BESTEFFORT_KILL" : El trabajo es de tipo Besteffort y fue matado por que un trabajo normal necesito del nodo.
18. "FRAG_JOB_REQUEST" : Alguien quiere eliminar un trabajo.
19. "CHECKPOINT" : la señal del realización de checkpoint se envió al trabajo.
20. "CHECKPOINT_ERROR" : OAR no pudo enviar la señal de creación de checkpoint al trabajo.
21. "CHECKPOINT_SUCCESS" : El sistemas envío la señal correctamente.
22. "SERVER_EPILOGUE_TIMEOUT": script epilogo del servidor esta en tiempo muerto.
23. "SERVER_EPILOGUE_EXIT_CODE_ERROR" : script epilogo del servidor no retorno cero.
24. "SERVER_EPILOGUE_ERROR" : No se encuentra el archivo script epilogo del servidor.
25. "SERVER_PROLOGUE_TIMEOUT" : Scrip prólogo del servidor esta en tiempo muerto.
26. "SERVER_PROLOGUE_EXIT_CODE_ERROR" : Scrip prólogo del servidor no retorno cero.
27. "SERVER_PROLOGUE_ERROR" :No se encuentra el archivo script prólogo del servidor.
28. "CPUSET_CLEAN_ERROR" : OAR no puede limpiar los archivos cpu set correctamente para un trabajo en el nodo remoto.
29. "MAIL_NOTIFICATION_ERROR" : El correo electrónico no pudo ser enviado.
30. "USER_MAIL_NOTIFICATION" : La notificación al usuario de correo no se puede realizar.
31. "USER_EXEC_NOTIFICATION_ERROR" : La notificación de ejecución de script de usuario no puede llevarse a cabo.
32. "BIPBIP_BAD_JOBID" : Error cuando se recuperaba información de un trabajo en ejecución (running job).
33. "BIPBIP_CHALLENGE" : Oar está configurado para separar trabajos cuando son lanzados en los nodos de cómputo y este retorna un bad challenge number.
34. "RESUBMIT_JOB_AUTOMATICALLY" : El trabajo se reenvío automáticamente.
35. "REDUCE_RESERVATION_WALLTIME" : La reserva del trabajo fue reducida
36. ''SSH_TRANSFER_TIMEOUT" : parte del script del nodo OAR era demasiado larga para su transferencia
37. "BAD_HASHTABLE_DUMP" : OAR transfirió una hashtable defectuosa.
38. "LAUNCHING_OAREXEC_TIMEOUT" : oarexec es demasiado largo para iniciarse a si mismo.
39. "RESERVATION_NO_NODE" : Todos los nodos se han detectado como malos para el trabajo de reserva.

Delante de cada uno de los errores listados en la tabla se responde si el administrador o el usuario están en capacidad de solucionar la falla que se presenta.

EVENTO O ERROR	ADMINISTRADOR	USUARIO
1. "PING_CHECKER_NODE_SUSPECTED"	SI	NO
2. "PING_CHECKER_NODE_SUSPECTED"	SI	NO
3. "EPILOGUE_ERROR"	SI	NO
4. "CANNOT_CREATE_TMP_DIRECTORY"	SI	NO
5. "CAN_NOT_WRITE_NODE_FILE"	SI	NO
6. "CAN_NOT_WRITE_PID_FILE"	SI	NO
7. "USER_SHELL"	SI	NO
8. "EXIT_VALUE_OAREXEC"	SI	NO
9. "SEND_KILL_JOB"	SI	SI
10. "LEON_KILL_BIPBIP_TIMEOUT"	SI	NO
11. "EXTERMINATE_JOB"	SI	NO
12. "WORKING_DIRECTORY"	SI	NO
13. ''OUTPUT_FILES"	SI	SI
14. "CANNOT_NOTIFY_OARSUB"	SI	NO
15. "WALLTIME"	SI	SI
16. "SCHEDULER_REDUCE_NB_NODES_FOR_RESERVATION"	SI	SI
17. "BESTEFFORT_KILL"	SI	SI
18. "FRAG_JOB_REQUEST"	SI	SI
19. "CHECKPOINT"	SI	SI
20."CHECKPOINT_ERROR"	SI	SI
21. "CHECKPOINT_SUCCESS"	SI	SI
22. "SERVER_EPILOGUE_TIMEOUT"	SI	NO
23. "SERVER_EPILOGUE_EXIT_CODE_ERROR"	SI	NO
24. "SERVER_EPILOGUE_ERROR"	SI	NO
25. "SERVER_PROLOGUE_TIMEOUT"	SI	NO
26. "SERVER_PROLOGUE_EXIT_CODE_ERROR"	SI	NO
27. "SERVER_PROLOGUE_ERROR"	SI	NO
28. "CPUSET_CLEAN_ERROR"	SI	NO
29. "MAIL_NOTIFICATION_ERROR"	SI	SI
30. "USER_MAIL_NOTIFICATION"	SI	NO
31. "USER_EXEC_NOTIFICATION_ERROR"	SI	NO
32. "BIPBIP_BAD_JOBID"	SI	NO
33. "BIPBIP_CHALLENGE"	SI	NO
34. "RESUBMIT_JOB_AUTOMATICALLY"	SI	SI

35. "REDUCE_RESERVATION_WALLTIME"	SI	SI
36. ''SSH_TRANSFER_TIMEOUT"	SI	NO
37. "BAD_HASHTABLE_DUMP"	SI	NO
38."LAUNCHING_OAREXEC_TIMEOUT"	SI	NO
39."RESERVATION_NO_NODE"	SI	SI

Tabla 4: *Errores presentes en OAR*

Con esta tabla presento una herramienta útil a administradores y usuarios del sistema OAR, para creación de estrategias de solución de errores propios de esta plataforma, también orienta a los usuarios sobre que errores pueden ser solucionados por ellos disminuyendo así la carga de administración de la plataforma.

Del análisis realizado de los diferentes errores presentes en OAR y la tabla anterior se puede observar que la mayoría de los eventos que son errores en el OAR, son ocasionados por mala configuración de la estructura, mas no durante la ejecución de un trabajo, estos errores se pueden prevenir realizando desde el comienzo una adecuada configuración de la infraestructura. El otro tipo de errores encontrados con frecuencia en el oar son de tipo factor humano, con esto se refiere a aquellos errores que no son ocasionados por fallas en la estructura de cómputo o una falla eléctrica o de infraestructura sino por una mala creación del código del trabajo enviado, siendo el mismo usuario el cual debe corregir este tipo de errores, por medio de la corrección del código fuente de su trabajo.

6.1.3 Selección de errores

Analizando las tablas anteriores se decide buscar estrategias de solución dentro de las fallas de red y las fallas físicas seleccionando las siguientes fallas:

- Perdida de un nodo por fallas en la estructura de comunicación.
  Esta falla se refiere al mal funcionamiento o pérdida de algún componente de comunicación router, swicth, cableado entre otros.
- Perdida del nodo y cálculos realizados debido a la finalización del tiempo cómputo del equipo.
  El tiempo de cómputo es el tiempo que el administrador asigna a la infraestructura para funcionar como un clúster al finalizar este los equipos retornan a su estado común.

- Perdida del trabajos por fallas en el suministro eléctrico del clúster.
  Falla que se presenta al no existir flujo eléctrico hacia los equipos que hacen parte del clúster ya sea por suspensión del servicio o por daños en la infraestructura eléctrica.

En busca de poder contar con un sistema que permita responder a estas fallas, se decide implementar una herramienta de creación de checkpoint que no solo permitiría responder a las fallas estructurales(físicas y de red), ya que con su implementación se brinda una solución a la limitación de tiempo de esta estructura, por que al guardar un estado de los trabajos antes de finalizar el tiempo de cómputo del nodo o el tiempo de ocio del equipo se podrá en un futuro reanudarlo a partir del estado en el que estaba al momento de crear el checkpoint, permitiendo que trabajos que necesiten mayor tiempo que el asignado para realizar cómputo, puedan ejecutarse en la plataforma basada en Computemode.

Dentro de las alternativas para la creación de checkpoints se encontraron:

- Herramienta Checkpoint/restart implementada en la versión 2.3 de OAR
- Implementar la herramienta DMTCP (checkpointing de multiprocesos distribuidos)
- Implementar BLCR (BERKLEY LAB CHECKPOINT / RESTART)

Durante la investigación se decidió implementar cada una de las posibilidades con el fin de comparar el modo de uso y el rendimiento de cada una de estas herramientas con las demás herramientas propuestas, a continuación se analizaran las diferentes alternativas con el fin de seleccionar la mas adecuada.

6.2 Análisis de las Herramientas de Checkpoint

Luego de definir los diferentes tipos de errores y fallas que se pueden presentar y realizar una categorización, se seleccionaron las fallas o errores a los cuales se les buscara un mecanismo de tolerancia encontrando que la mejor opción fue la realización de checkpoints para ofrecer una solución. A continuación se realiza el análisis y prueba de las diferentes opciones de creación de checkpoints realizando un análisis más profundo en su funcionamiento y modo de implementación.

6.2 .1 Sistema Checkpoint/restart implementado en OAR

Comenzando con la selección de mecanismos de restauración de procesos, se realizo el análisis de las diferentes opciones, en primer lugar se analizo el checkpoin/restart de la herramienta manejadora de colas OAR, la cual está presente en su versión 2.3, lanzada durante el desarrollo de este proyecto por lo que antes no había sido tomada en cuenta, se toma como primera opción ya que Computemode incluye OAR como manejador de colas y la alteración de la infraestructura seria mínima.

Se investigo la forma de usar este sistema encontrándose que el comando de envió de trabajos con checkpoint era el siguiente:

oarsub -l “nodes= 2 , walltime = 01:00:00” - -checkpoint=300 ./ejecutable

Donde:

- “nodes=” Indica el numero de nodos a usar.
- “walltime =” Hace referencia al tiempo de cómputo.
- “--checkpoint=” tiempo en segundos para realizar el checkpoint.

Luego de realizar la ejecución de un trabajo mediante el uso de este comando se observo que cumplidos 300 segundos el trabajo no creaba ningún archivo o notificaba de alguna acción, pero faltando 300 segundos para terminar el tiempo de cómputo, el trabajo finalizaba abruptamente, lo que no era normal, cabe señalar que el ejecutable ya había sido probado con anterioridad obteniendo un optimo resultado.

Se realizo un análisis de las librerías que componen OAR para saber en cuales de estas se manejaba la realización de checkpoints, con el objetivo de verificar su funcionamiento, las librerías que contenían código que intervenían durante el proceso son las siguientes:

1. oardel
2. oarexec
3. oar_iolib.pm
4. oarsub
5. oar_Tools.pm
6. sarko

De las diferentes librerías se comenzó por la librería oarsub ya que esta librería es la que se encarga del envío de trabajos y es la encargada de tomar los datos ingresados por el usuario entre estos el tiempo de checkpoint y según los diferentes llamados a funciones se recorrieron todas las librerías antes mencionadas, durante el análisis se llego a la librería sarko la cual contiene el siguiente código.

```
elsif (($job->{checkpoint} > 0) && ($current >= ($start+$max-$job->{checkpoint}))){
    # OAR must notify the job to checkpoint itself
    oar_debug("[sarko] Send checkpoint signal to the job $job->{job_id}\n");
    # Retrieve node names used by the job
    my @hosts = iolib::get_job_current_hostnames($base,$job->{job_id});
    iolib::add_new_event($base,"CHECKPOINT",$job->{job_id},"User oar (sarko) requested a checkpoint on the job $job->{job_id}");
    my $str_comment;
    my @exit_codes;
    # Timeout the ssh command
    eval {
      $SIG{ALRM} = sub { die "alarm\n" };
      alarm(oar_Tools::get_ssh_timeout());
      @exit_codes = oar_Tools::signal_oarexec($hosts[0],$job->{job_id},"SIGUSR2",1,$base, $Openssh_cmd);
      alarm(0);
    };
```

En la primera línea se observa que el fragmento de código solo se ejecutara si el valor de checkpoint es mayor que cero es decir si se introdujo la opción de checkpoint igual a un número mayor que cero, y también condiciona su ejecución a realizarse faltando esa misma cantidad de segundos para finalizar el tiempo de cómputo, indicando también que solo se ejecutara una vez durante toda la ejecución, el segundo fragmento resalta do en negrilla muestra el llamado ala función signal_oarexec de la librería oar_Tools esta función se analiza a continuación .

Se busco en cada una de las librerías, alguna acción que mostrara la creación de un archivo de checkpoint, pero no se encontró ninguna sentencia que realizara esto, en cambio dentro de la librería oar_Tools en la sub función signal_oarexec llamada por sarko (fragmento de código anterior), se encuentran las siguientes instrucciones

```
foreach my $p (@cmd_opts){
```

```
        $cmd[$c] = $p;$c++;
    }
```

Creación de comando que se desea ejecutar, en este caso el comando kill a cada proceso

```
    $cmd[$c] = "-x";$c++;
    $cmd[$c] = "-T";$c++;
    $cmd[$c] = $host;$c++;
    $cmd[$c] = "bash -c 'test -e $file && PROC=\$(cat $file) &&
kill -s CONT \$PROC && kill -s $signal \$PROC'";$c++;
```

Verificación de conexión ssh hacia los procesos.

```
$ssh_pid = fork();
            if ($ssh_pid == 0){
                exec({$cmd_name} @cmd);
                warn("[ERROR] Cannot find @cmd\n");
                exit(-1);}
```

Del análisis de este fragmento de código echo en perl se observo que se pide finalizar con los procesos que hacen parte del trabajo al que se le ingreso la opción de checkpoint, esto explica el comportamiento observado durante las pruebas en donde los trabajos al indicárseles la opción de checkpoint a los n segundos, faltando n segundos para cumplirse su tiempo de cómputo los procesos que pertenecían al trabajo finalizaban y junto a los procesos la aplicación.

Por el análisis de las diferentes librerías y el resultado arrojado por las pruebas se decidió.

Que el sistema Checkpoint/restart de OAR no era un sistema para guardar estados de los trabajos, para un futuro reinicio de estos, sino que era una simple señal de finalización de trabajos faltados n segundos indicados por el usuario para finalizar el tiempo de cómputo. Como lo que buscamos es un mecanismo de realización de checkpoints que permita reiniciar un trabajo después de haberse presentado una falla, descartamos este mecanismo por no cumplir con los requisitos que buscamos.

Se deja claro que el OAR no tiene un mecanismo de realización de checkpoint, pero sus librerías permiten la integración de un mecanismo de checkpointing al modificar el fragmento de código que se encarga de realizar la finalización de los trabajos para que en lugar de realizar esta acción, introduzcan alguna acción de creación de checkpoint o integración a alguna herramienta que realice esta acción.

6.2.2 Herramienta DMTCP (checkpointing de multiprocesos distribuidos)

DMTCP (Distributed MultiThreaded Checkpointing) se tomo como segunda opción para la realización de checkpoints puesto que es una herramienta que no requiere ningún cambio en el equipo, debido a que esta funciona a nivel de usuario comportándose como si fuese una aplicación, se opto por realizar la instalación de las herramientas a partir de la descarga del paquete en extensión .tar.gz, ya que esta contiene las fuentes del software y permiten una configuración más profunda que las otras versiones al momento de la instalación, para la instalación es necesario seguir los siguientes pasos.

1) descargamos la versión .tar.gz mas reciente de la página oficial del proyecto: http://dmtcp.sourceforge.net/downloads.html
2) Extraemos los archivos: tar xzvf dmtcp-x.x.tar.gz y luego nos ubicamos en el directorio extraído.
3) Dentro del directorio ingresamos los comandos: ./configure y make

Con lo anterior finaliza la instalación de DMTCP, podemos observar que de cierta manera su instalación es sencilla pero es importante saber que el software queda instalado en la misma carpeta en que se realiza la compilación, debido a esto el directorio de compilación debe ser un directorio accesible para todos los usuarios, lo que es muy difícil de hacer para un clúster basado en Computemode, ya que cada usuario tiene su propio espacio, con carpetas y archivos propios.

6.2.2.1 Creación de checkpoint usando DMTCP

Luego de su instalación se prosigue con el análisis del proceso de creación de checkpoint. Estos son los pasos para realizar el checkpoint.

Ubicado dentro del directorio donde se instalo el software se deben seguir los siguientes pasos, tenga en cuenta que para la ejecución del coordinador *dmtcp_coordinator* y el *dmtcp_checkpoint* se realizan en consolas o terminales diferentes:

1) En una terminal inicie el coordinador de dmtcp que se encuentra en el directorio de compilación mediante el comando.

dmtcp_coordinator

2) En otra terminal(es), dentro del directorio de compilación ejecute el comando: *dmtcp_checkpoint*, el programa encargado de realizar los checkpoints se conectara con el coordinador expacifico para *DMTCP_HOST* and *DMTCP_PORT*.

   dmtcp_checkpoint ./mi_programa

3) En estos momentos corriendo la aplicación en la(s) terminal(es) abierta(s) con *dmtcp_checkpoint*, diríjase a la terminal donde se ejecuta el coordinador y introduciendo la letra c el creara un checkpoint de la aplicación con el nombre dmtcp_restart_script.sh.
   dmtcp_command –c

4) Para reiniciar los trabajos solo debe usar el archivo dmtcp_restart_script.sh creado por el dmtcp_coordinator . Opcionalmente puede modificar este script para migrar los procesos a equipos diferentes, con el fin de realizar un reinicio estándar, la secuencia de comandos debe ser editada parapara ejecutar el proceso deseado en el primer plano de una terminal.

Tomando en cuenta los pasos anteriores se observa que esta alternativa, no va de acuerdo a las políticas de privacidad de los usuarios, para los que usar una carpeta pública para correr el trabajo, generaría desconfianza en su uso, junto con esto los pasos para crear el checkpoint son muy complicados y diversos lo que supondría más trabajo para el usuario y un previo adiestramiento en su uso para poder realizar un correcto manejo, y en realidad se busca que el proceso de creación sea lo más transparente posible para el usuario de la plataforma.

Por la complicación en su uso esta alternativa se deja como opción pero no se implementara, ya que cumple con su función, pero mediante un mecanismo difícil de usar.

6.2.3 BLCR (BERKLEY LAB CHECKPOINT / RESTART)

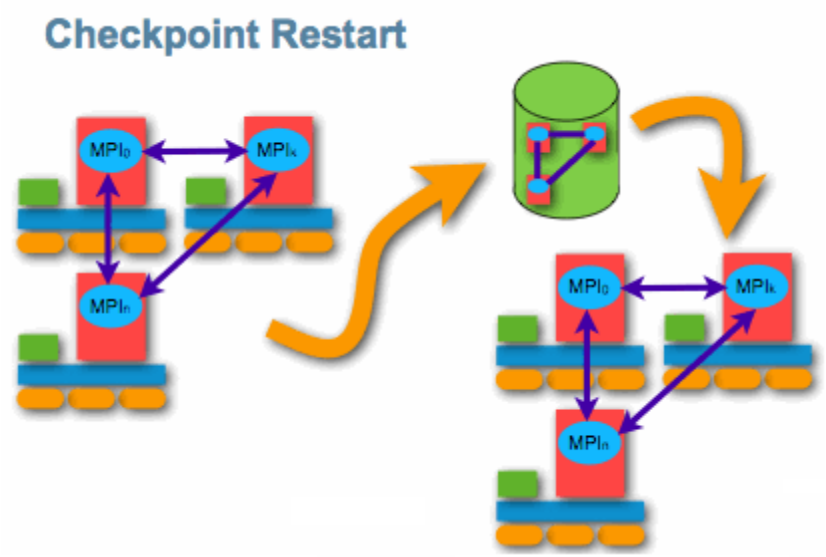


Figura 4: Funcionamiento Checkpoint[19]

Luego de analizar las opciones anteriores sin obtener resultados óptimos se decide implementar la herramienta BLCR, esta es una herramienta que funciona a nivel del sistema por lo que es necesario compilarla usando los códigos fuente de la versión del kernel en el cual se usara, luego realizar las configuraciones necesarias para integrarla a alguna herramienta como MPI, OPENMPI, ó MPICH a continuación se describe los pasos que se siguieron para su instalación e integración.

6.2.3.1 KERNEL

Recordemos que BLCR es una herramienta de nivel de sistema por lo cual es necesario compilarla usando las fuentes del kernel. Para realizar una compilación de BLCR se investigó que tipo de kernel unix soportaba, dentro de la página oficial del

[19] Imagen tomada de la conferencia HPC Trends and Virtualization de sarrollada por Josh Simons Sun Distinguished Engineer

proyecto se encontró que BLCR en su versión 0.8.2 cuenta con soporte hasta la versión 2.6.30 de kernel unix, luego de realizar una compilación de las diferentes versiones de kernel desde la 2.6.26 hasta la 2.6.30 se descubrió que su funcionamiento no presentaba problemas con el Computemode hasta la versión 2.6.29 o anteriores, pero al momento de usar la estructura de cómputo existente se decidió trabajar con la versión 2.6.26-2, esto se debió a que su desarrollo fue basado en esta versión del kernel lo que le da mayor soporte a sus componentes. Los pasos de compilación de un kernel se encuentran en el anexo A.

6.3 Instalación de BLCR

La investigación realizada sobre la herramienta permitió conocer que BLCR cuenta con un soporte para las aplicaciones más comunes LAM/MPI, MPI, MPICH, OPENMPI, entre otras. Esto quiere decir que se pueden integran con dichos sistemas durante su instalación, pero realizar esta integración requiere de seguir las indicaciones que se darán a continuación, este proceso se obtuvo, luego de realizar varias instalaciones hasta encontrar la forma correcta de realizarlas.

Se decidió realizar la integración de BLCR con la herramienta OPENMPI, dentro del anexo B se encuentra una explicación de cómo realizar esta integración usando LAM/MPI, esta decisión se tomó basándose en el estado de cada uno de los proyectos, con esto se quiere hacer referencia a la continua mejora y soporte que se realiza en cada una por parte de sus desarrolladores, ya que LAM/MPI no ha mostrado nuevas versiones o actualizaciones de la herramienta se considero mejor el uso de OPENMPI la cual cuenta con un estado mas actualizado, las razones se explican más a fondo en el anexo B de instalación de LAM/MPI.

Para la instalación de BLCR se deben tener en cuenta las siguientes indicaciones.

1. Debe copiar las fuentes de compilación del kernel en la carpeta /usr/src para no tener problemas durante la instalación.
2. Seleccione un prefix de manera que tenga los archivos de BLCR en una carpeta independiente y no se mezclen con los archivos de otras aplicaciones, esto se sugiere ya que se observo errores de funcionamiento al realizar la instalación en el directorio por defecto /usr/local, el prefix permite de manera más sencilla la inclusión del programa a los nodos.
3. Se recomienda que el kernel que este corriendo en el momento de la instalación de BLCR sea en el que BLCR usará.

6.3.1 Instalación en el servidor de Computemode

El clúster basado en Computemode tiene una particularidad muy importante, la instalación de aplicaciones en el servidor se realiza mediante métodos diferentes a la instalación de aplicaciones en los nodos. Para la instalación de BLCR dentro del servidor se siguieron los pasos que se listan a continuación:

1) Se realizo la compilación del kernel que se utilizaría tanto en el servidor como en los nodos, teniendo el cuidado de que las fuentes de este quedaran instaladas dentro del archivo

   #/usr/src.

2) Se descargo el paquete BLCR en su versión blcr-0.8.2.tar.gz
3) Se extrajo el contenido se ingreso en la carpeta y se desarrollo la configuración usando los siguientes comandos.

   # mkdir builddir

   El anterior comando crea una carpeta la cual usaremos para albergar los archivos de configuración del BLCR, paso a seguir ingresamos en la carpeta y procedemos con la configuración

   # cd builddir
   #../configure **- -prefix=/opt/blcr**

Al momento de realizar la configuración especificamos el directorio de instalación, esto lo hacemos para evitar posibles conflictos con otras aplicaciones que se encuentren instaladas en el directorio /usr/local, durante la investigación se realizo la instalación sin especificar el directorio de instalación lo que produjo errores durante el proceso de creación y reinicio de checkpoints, que es el directorio de instalación por omisión, también esta acción facilitara la instalación del BLCR en los nodos

4) Se realizó la instalación mediante los comandos
   # make
   # make install

5) Luego de la instalación debe hacerse la configuración de los diferentes PATH y una inclusión de los módulos al kernel, para realizar esta inclusión se presentan dos opciones, la primera consistía en agregarlos a la imagen del kernel que se creó anteriormente usando el mismo proceso utilizado para compilar el kernel, la segunda consistía programar la inclusión de los módulos durante la carga del sistema, a diferencia de la primera opción esta no requiere modificaciones de bajo nivel en el sistema, que puedan afectar el funcionamiento de la imagen desplegada en el servidor y los nodos. Se escogió la segunda opción por lo antes explicado cuya implementación consta de los siguientes pasos.

   a) Dentro de la carpeta /etc/init.d del servidor creamos el archivo levantar BLCR, este es un pequeño script que contiene las siguientes líneas de código.

```
#!/bin/sh
#Proceso de arranque de modulos BLCR
/sbin/insmod /usr/local/lib/blcr/2.6.26.2/blcr_imports.ko
/sbin/insmod /usr/local/lib/blcr/2.6.26.2/blcr.ko
```

   b) Estas instrucciones cargaran los módulos de BLCR pero para que se realice su ejecución en el momento de iniciar el sistema se deben dar permisos de ejecución y introducirlo dentro del listado de aplicaciones de inicio utilizando los siguientes comandos.
      # chmod 700 levantar BLCR

      # update-rc.d levanterBLCR defaults

   Para la configuración de los PATH de Linux se incluyeron cambios dentro del archivo profile Que es el archivo que usa el sistema para cargar el los diferente PATH y se encuentre ubicado en la carpeta etc en el directorio raíz del sistema, a continuación se muestra partes pertenecientes al archivo profile resaltando en negrilla las modificaciones que se le realizaron.

```
# /etc/profile: system-wide.profile file for the Bourne shell
(sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1),...).

if [ "`id -u`" -eq 0 ]; then

PATH="/usr/local/sbin:/opt/blcr/bin:/usr/sbin:/usr/bin:/sbin:/
bin"
else

PATH="/opt/blcr/bin:/usr/bin:/bin:/opt/openmpi/bin:/usr/games"
```

El fragmento anterior se realiza la inclusión de las carpetas de binarios tanto de BLCR como OPEM MPI al PATH de sistema

LD_LIBRARY_PATH=$LD_LIBRARY_PATH**:/opt/blcr/lib:/opt/openmpi/lib**:/usr/lib

En este fragmento se realiza la inclusión de las librerías tanto de BLCR como OPEM MPI al LD_LIBRARI_PATH de sistema

Estos cambios son necesarios para que al momento de iniciar el sistema conozca la ubicación de los ejecutables y librerías de las dos aplicaciones.

Ya instalado BLCR en el servidor es de esperarse que se proceda con la instalación de BLCR en los nodos, pero antes de realizar la instalación en los nodos se debe realizar la integración con la herramienta OPENMPI ya que este proceso introduce cambios en los archivos de BLCR, esto se observó durante los diferentes procesos de instalación realizados. Teniendo esto presente se explica a continuación la instalación de OPENMPI dentro del servidor Computemode.

6.3.2 Instalación de OPENMPI en el servidor de Computemode

Como la realización de checkpoints mediante el uso de BLCR es una propiedad que se aplica solo dentro de los equipos que lo contengan, las opciones de configuración de BLCR vienen deshabilitadas por defecto, es por esto que se deben habilitar en el momento de la configuración de la instalación de OPENMPI, en el siguiente

procedimiento paso a paso se muestra la correcta configuración y instalación de OPENMPI integrándola con BLCR este proceso se obtuvo como resultado de la investigación realizada, mediante la utilización de distintas configuraciones, instalación y prueba de la herramienta.

1) Para esta instalación se usara la versión openmpi-1.4.3 que a la fecha de realización de este proyecto es la última versión estable publicada, ubicados en el archivo en el que se ubica el archivo openmpi-1.4.3.tar.gz procedemos a extraerlo y ingresar en la carpeta mediante los siguientes comandos.
   # tar xvzf openmpi-1.4.3.targz
   # cd openmpi-1.4.3

2) Una vez dentro del directorio realizamos la configuración, esta parte de la instalación se recomienda hacer la de la forma que se indica, ya que el realizar cambios al comando de configuración puede hacer que la aplicación no funcione correctamente o en el momento de instalación está presente errores.

   El comando configuración es el siguiente.
   ./configure --prefix=/opt/openmpi --enable-ft-thread --with-ft=cr --enable-mpi-threads --enable-opal-multi-threads --with-blcr=/opt/blcr/ --with-blcr-libdir=/opt/blcr/lib

   Donde:
   - La opción --prefix sirve para indicar el lugar donde se instalara el openmpi
   - La opción --with-blcr indica el directorio de instalación del BLCR
   - La opción –with-blcr-libdir indica el directorio donde se encuentran las librerías de BLCR
   - Las demás opciones son para activación de propiedades necesarias para la realización de los checkpoints.

   Debe tenerse en cuenta que los directorios indicados en los comandos --with-blcr y - -with-blcr-libdir, hacen referencia a los directorios seleccionados durante la instalación de BLCR.

La opción *enable-mpi-threads* y *enable-opal-multi-threads* no van juntas la primera fue remplazada en nuevas versiones de OPENMPI por la segunda, su uso depende de la versión de OPENMPI que este configurando, para evitar confusiones durante la instalación ingresar las dos, la configuración descartara la que no conozca.

3) Ya realizada la configuración se debe instalar mediante el comando
   # make all install

Luego de haber realizado la instalación de OPENMPI se han introducido cambios dentro de los archivos de instalación de BLCR y ya contamos con la carpeta que contiene la instalación de OPENMPI, El siguiente paso es integrar las aplicaciones a los nodos.

6.3.3 Instalación en los nodos de Computemode

Luego de realizada la instalación en el servidor se procedió a modificar el directorio /fsdiskless el cual es el que se envía a los diferentes nodos de forma que en cada nodo que se fuese agregando al Computemode contara con la instalación de BLCR y OPENMPI con las mismas cualidades de las instaladas en el servidor, para realizar esto en caso de no haber indicado el directorio de instalación de BLCR y haberla realizado en el directorio por defecto se deben agrupar los archivos que forman parte de la aplicación en una carpeta, la cual contendrá la instalación de BLCR las carpetas que se deben agrupar son:

- carpetas /usr/local/bin
- carpetas /usr/local/etc
- carpetas /usr/local/include
- carpetas /usr/local/lib
- carpetas /usr/local/man
- carpetas /usr/local/share

Estas carpetas se integraron dentro del directorio /opt/blcr que hace parte del directorio /fsdiskless, en el caso de realizar el prefix como se indico durante la instalación del BLCR solo se debe copiar la carpeta de instalación dentro del

directorio, también debe copiarse la carpeta que contiene la instalación de OPENMPI en este directorio.

Luego de realizar la copia de BLCR Y OPENMPI al directorio correspondiente, se realizaron modificaciones al archivo Buildroot.sh el cual se encarga de cargar las aplicaciones, al archivo se le añadieron las siguientes instrucciones.

1) Instalación de OPENMPI al nodo
   cp -ra $FUENTE/opt/openmpi /opt/
   cp -ra $FUENTE/usr/bin/iperf /usr/bin/

2) Instalación de BLCR
   cp -ra $FUENTE/opt/blcr /opt

A diferencia del servidor de Computemode en el cual se creó un script que realizara la carga de los módulos de BLCR, en el caso de los nodos se añadió al archivo Buildroot.sh las siguientes líneas para que durante la ejecución de este script se realizara la cargara los módulos de BLCR.

3) Carga de los modulos de BLCR
   /sbin/insmod /opt/blcr/lib/blcr/2.6.26.2/blcr_imports.ko
   /sbin/insmod /opt/blcr/lib/blcr/2.6.26.2/blcr.ko

Hasta el momento hemos realizado la copia e instalación de OPENMPI Y BLCR en los nodos, luego debemos ingresar sus carpetas a los diferentes PATH del sistema, para realizar esto normalmente se introducen estos usando el archivo profile ubicado en el directorio /etc del sistema de archivos, este proceso fue el utilizado en el servidor, para los nodos no se pudo realizar este proceso ya que al momento de intentar crear el archivo profile dentro de la carpeta /fsdiskless/etc este archivo no se guardaba con los cambios incluidos, para solucionar este inconveniente se opto por crear el archivo profile2, el cual es igual al archivo profile del servidor, para luego durante la carga del sistema de archivos del nodo remplazar la información contenida dentro del archivo profile por la contenida dentro del archivo profile2, para lograr esto se realizo una nueva modificación al archivo Buildroot.sh.

cp -r $FUENTE/etc /

cp /etc/profile2 /etc/profile

En la primera instrucción se realiza el remplazo del directorio etc de los nodos por el directorio etc contenido dentro del directorio fsdiskless del servidor OAR. Paso a seguir en la segunda instrucción se realiza la copia del contenido del archivo profile2 al archivo profile, con esto queda realizado la parte de los PATH del sistema.

Completando todos los pasos explicados para la instalación de las herramientas BLCR y OPENMPI (OPENMPI con las propiedades de checkpoint y la integración de BLCR activadas y configuradas), dentro del servidor y los nodos el sistema de tolerancia a fallas debe estar funcionando. En la sección que sigue se explica el uso del sistema.

6.4 Creación de checkpoints usando BLCR

Para realizar un checkpoint utilizando BLCR se debe de cumplir dos etapas por parte del usuario, la primera de estas etapas consiste en el lanzamiento o ejecución del trabajo, la segunda etapa, es la petición de realización del checkpoint, esta petición de checkpoint se realiza mediante la ejecución de dos comandos uno que le proporciona al usuario el PID del proceso que el usuario ejecuta, el otro comando es en el cual usando el PID se envía la petición de realización de checkpoint del trabajo. Buscando que para el usuario sea necesario realizar únicamente la primera etapa, el envío o ejecución de trabajos, se automatizó la segunda etapa agregando un cron a los nodos el cual realiza checkpoint periódicos de las aplicaciones, esto no quiere decir que todos los nodos luego de cierto tiempo realicen una copia de todos los procesos que en ellos se ejecutan. El script solo pide al nodo que ejecuto el comando mpirun realizar el checkpoint de este proceso, en caso de que el nodo no haya sido el que ejecutó el comando mpirun no realizara ninguna acción el cron que se instalo, a continuación se muestra el código del ejecutable y los pasos para que este se ejecute periódicamente.

El siguiente texto es el contenido del archivo share.sh el cual contiene los comandos necesarios para realizar la petición de creación de checkpoint y obtención del PID del proceso.

```
#!/bin/bash
```

```
#seleccion de usuario y id del proceso, para luego realizar
checkpoint
usuario=$(ps ax -o user,pid,command | grep mpirun |grep -v
grep|cut -d " " -f1)
cd /home/$usuario
idproceso=`ps ax -o pid,user,command | grep mpirun |grep -v
grep|cut -d " " -f2 `
echo "#!/bin/bash" > /opt/ts.sh
echo   "/opt/openmpi/bin/ompi-checkpoint   $idproceso"   >>
/opt/ts.sh
chmod +x /opt/ts.sh
su $usuario -c '/opt/ts.sh'
```

El funcionamiento del share.sh es el siguiente obtiene el usuario que realizo el envío del proceso luego se realiza la creación del script ts.sh que contiene los comandos de petición de checkpoint, paso a seguir se ejecuta el script ts.sh con las propiedades del usuario obtenido. Como se busca que esta acción sea realizada cada cierto tiempo sin necesidad de que el usuario deba digitar algún comando, este ejecutable se añadió como un cron del sistema, para realizarlo se modifico el archivo crontab de la carpeta fsdiskless, para que los nodos realizaran la ejecución del archivo share.sh cada n minutos, a continuación se muestra el archivo resaltando en negrilla el código que se le incluyo.

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this
file
# and files in /etc/cron.d. These files also have username
fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/u
sr/bin

# m h dom mon dow user command
17  *    *  *  *    root    cd  /  &&  run-parts  --report
/etc/cron.hourly
25 6  * * *  root  test -x /usr/sbin/anacron || ( cd / &&
run-parts --report /etc/cron.daily )
47 6  * * 7  root  test -x /usr/sbin/anacron || ( cd / &&
run-parts --report /etc/cron.weekly )
```

```
52 6   1 * *   root   test -x /usr/sbin/anacron || ( cd / &&
run-parts --report /etc/cron.monthly)
*/15 *   * * *   root   /opt/shared.sh#
```

Después de aplicados estos cambios el usuario solo debe enviar su trabajo y los nodos se encargan de ejecutarlo y realizarle un checkpoint cada vez que se cumplan n minutos, en este caso cada 15 minutos, se realizara la ejecución de share.sh.

El comando de envío de trabajos es el siguiente

/opt/openmpi/bin/mpirun -np 2 -machinefile machines -am ft-enable-cr Ejecutable

Donde:
-np indica el numero de procesos en que se divide el trabajo.
-machinefile indica el archivo que contiene el listado de los nodos en que se ejecutara el trabajo.
-am ft-enable-cr se usa para habilitar la opción de creación de checkpoint para el ejecutable del trabajo.

Se debe tener en cuenta que el directorio que se crea y se usa como checkpoints y que es generado de manera transparente para el usuario, se crea dentro de la carpeta que le pertenece al usuario, es decir dentro del directorio /home/$usuario, $usuario hace referencia al nombre del usuario que ejecutó el trabajo, por esto para reiniciar el trabajo el usuario debe conectarse a la plataforma iniciar sesión en algún nodo y lanzar el trabajo con el siguiente comando.

ompi-restart -hostfile maquinas nombre del directorio de checkpoint

La opción –hostfile no es obligatoria y sirve para indicar los nodos donde se relanzara el trabajo, el usuario debe tener en cuenta que el nodo en que lanzo por primera vez el trabajo debe estar disponible, por esto se recomienda, reiniciar un trabajo luego de iniciar sección en el nodo en que se había realizado la ejecución.

7. PRUEBAS Y RESULTADOS

Las siguientes pruebas las realizamos con el fin de verificar el cumplimiento de tolerancia a las fallas seleccionadas; verificar el funcionamiento de las tareas programadas; la realización correcta de los checkpoint y la correcta culminación de los trabajos que se reenviaban utilizando los checkpoints creados. Para la realización de las diferentes pruebas del mecanismo de tolerancia a fallas se tuvo en cuenta el listado de fallas seleccionadas anteriormente, estas fallas fueron las siguientes:

- Perdida de un nodo por fallas en la estructura de comunicación.
  Esta falla se refiere al mal funcionamiento o pérdida de algún componente de comunicación router, swicth, cableado entre otros.
- Perdida del nodo y cálculos realizados debido a la finalización del tiempo cómputo del equipo.
  El tiempo de cómputo es el tiempo que el administrador asigna a la infraestructura para funcionar como un clúster al finalizar este los equipos retornan a su estado común.

- Perdida del trabajos por fallas en el suministro eléctrico del clúster.
  Falla que se presenta al no existir flujo eléctrico hacia los equipos que hacen parte del clúster ya sea por suspensión del servicio o por daños en la infraestructura eléctrica.

7.1 Perdida de un nodo por fallas en la estructura de comunicación

Para analizar esta falla se realizo el envío de un trabajo a dos nodos y durante el desarrollo de la ejecución se le retiro el cable de red a uno de los nodos esto se realizo para simular la pérdida del nodo ya sea por daños en el cable o un dispositivo de red.

Primero iniciamos sesión en un nodo y realizamos el envío de un trabajo desde este nodo usando un ejecutable que calcula el área en el intervalo de 0 hasta 15 dividido en 1500.000.000 secciones.

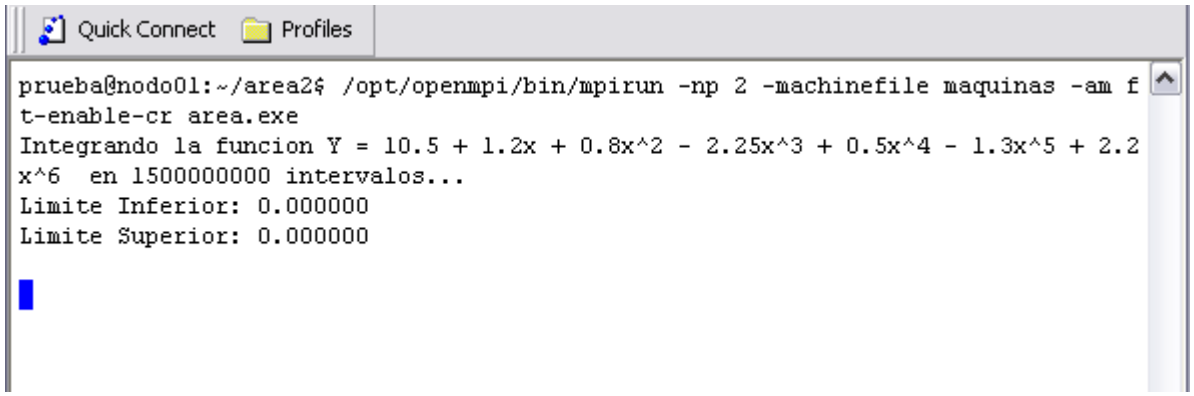


Figura 5: Envió de trabajo prueba1

En la figura siguiente se observa el comando watch ls –l dentro de la carpeta del usuario este comando sirve para monitorear cada dos segundos la carpeta realizando el comando ls –l el cual lista su contenido y propiedades de los archivos

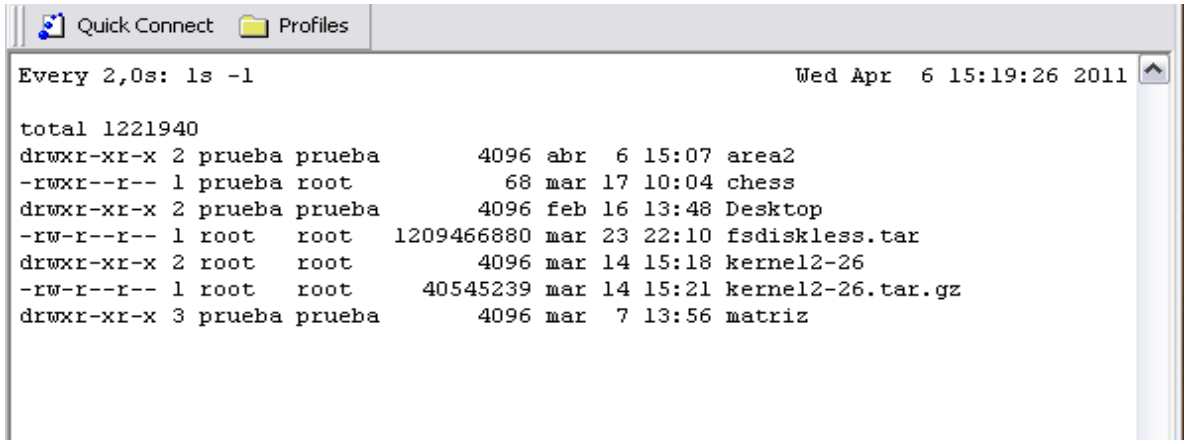


Figura 6: Primer comando watch ls –l prueba1

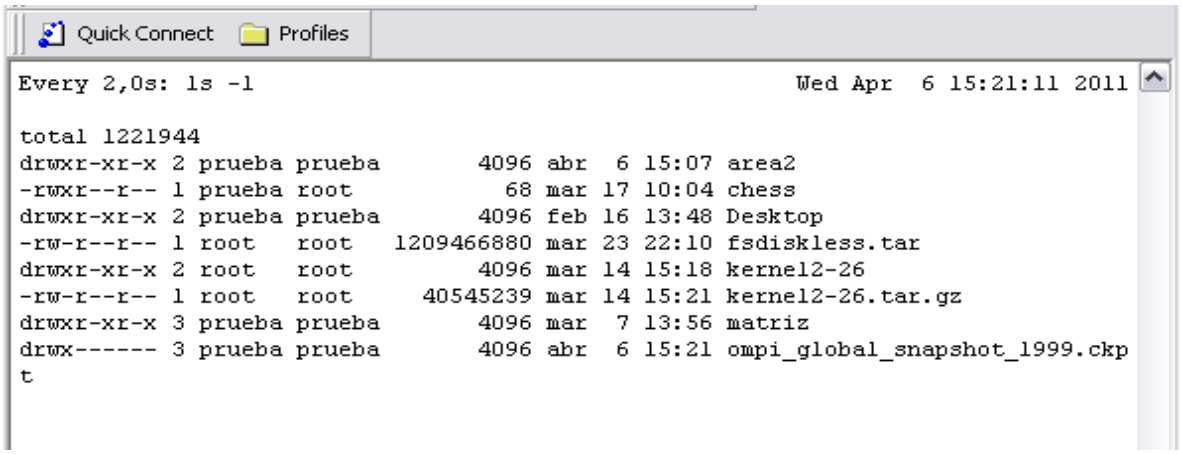


Figura 7: Segundo comando watch ls –l prueba1

Las dos figuras muestran el comando watch ls –l el cual usamos para verificar que la realización automática de checkpoints se esté realizando, pasados aproximadamente 3 minutos se creó la carpeta ompi_global_snapshot_1999.ckp, la cual contiene los checkpoints de los diferentes procesos, esto indica que la realización automática de checkpoins se encuentra funcionando, para la realización de las pruebas se configuro la realización de checkpoint cada 3 minutos, con el fin de comprobar si cada 3 minutos se realiza el checkpoint se realizo el comando watch ls –l dentro de la carpeta de checkpoint dando el siguiente resultado.

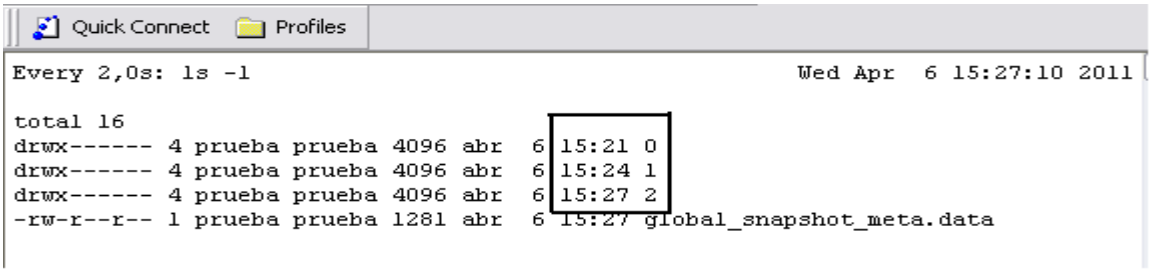


Figura 8: Intervalos de tiempo en la creación de checkpoints prueba1

Se puede con esto observa que el intervalo de creación entre un checkpoint y otro es de 3 minutos, lo que nos indica que la tarea programada, se está realizando según las indicaciones dadas.

Paso a seguir se provocó la falla del sistema retirando el cable de red de uno de los nodos incluidos en el archivo maquinas, el trabajo continua realizándose hasta el momento en que se pide la recolección de datos de los diferentes nodos.

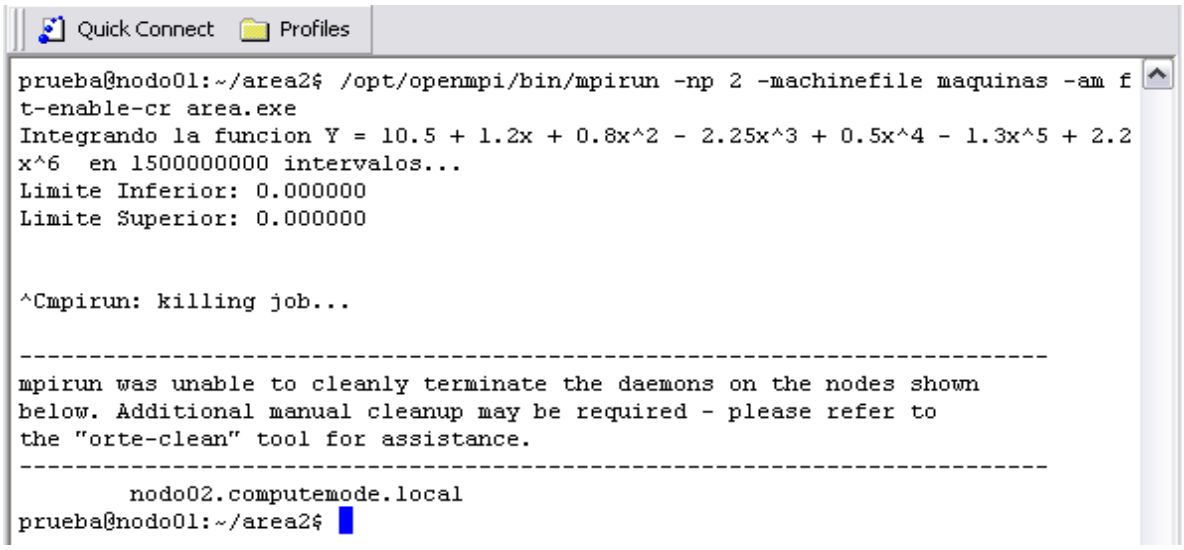
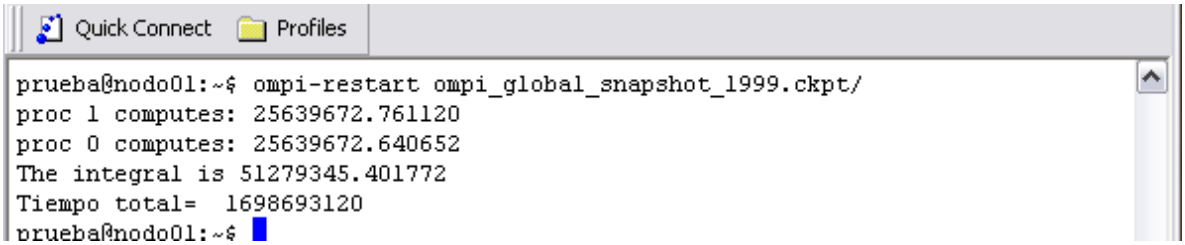
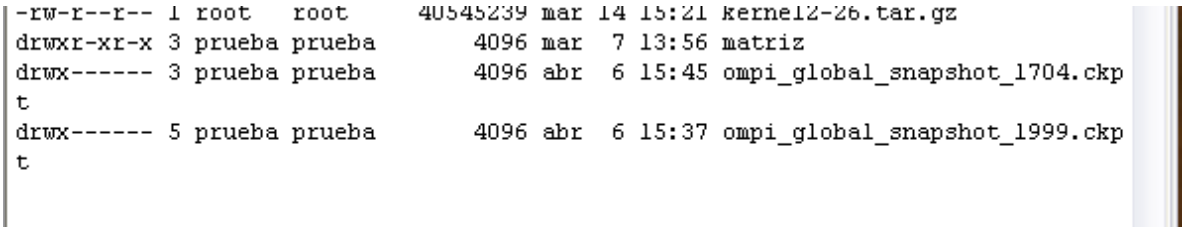


Figura 9: Perdida nodo prueba1

Se observa que la comunicación con el nodo02 se perdió por esto el trabajo finaliza, luego de un tiempo se reconecta el nodo lo que en el caso de la vida real se refiere a la sustitución o reparación del nodo

Luego de la reparación del nodo se realizo el reenvío del trabajo, en la siguiente figura se muestran tres estados el reenvió del trabajo, como culmina y como se generan automáticamente los checkpoints del proceso que se relanzo.





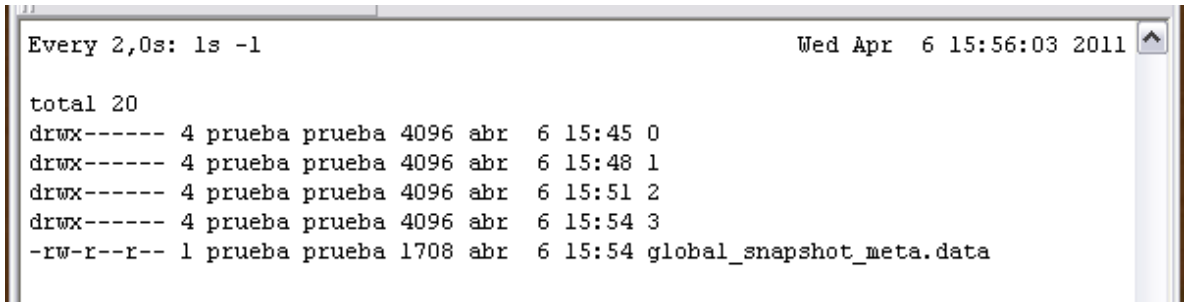


Figura 10: Reenvío de trabajo prueba1

Esto se culmina la prueba de pérdida de nodo, obteniendo un resultado óptimo de respuesta por parte del mecanismo de tolerancia a fallas al error generado.

7.2 Perdida del nodo y cálculos realizados debido a la finalización del tiempo cómputo del equipo.

La prueba realizada anteriormente sirve para demostrar que el sistema responde también a esta falla, esto se debe a que la finalización del tiempo de cómputo es un error que se presenta debido a la pérdida de nodos de cómputo (igual que en el caso anterior), por el retorno de estos a sus actividades comunes. Debido a que en el servidor se guardan los diferentes checkpoints, solo se debe esperar el momento en que los nodos se encuentren nuevamente en tiempo de cómputo, y por medio del uso de los archivos de checkpoint realizar el reenvío de los trabajos que no finalizaron y así continuar con su ejecución.

7.3 Perdida de trabajos por fallas en el suministro eléctrico del clúster.

Con esta prueba se busco ver el funcionamiento de los checkpoint después de perder no solo los nodos sino también el servidor donde se almacenan los checkpoints, la prueba permite examinar la respuesta del mecanismo de tolerancia a fallas a los daños en la infraestructura eléctrica o a la suspensión del servicio.

Primero se realizo la ejecución del trabajo y se le permitió realizar varios checkpoints transcurridos aproximadamente 12 minutos se suspendió el servicio de corriente de todos los equipos.

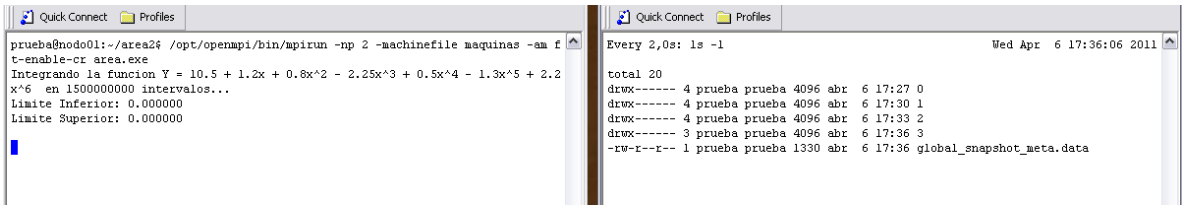


Figura 11: Envío prueba 2

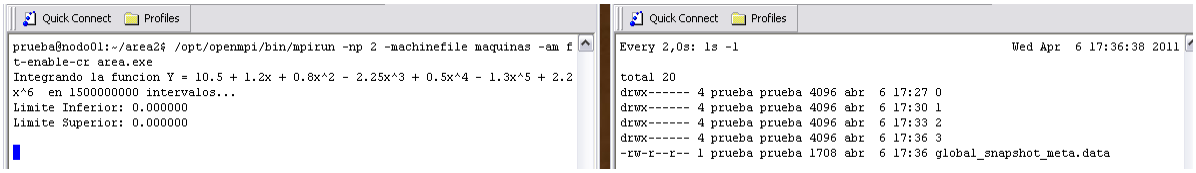


Figura 12: Checkpoints prueba2

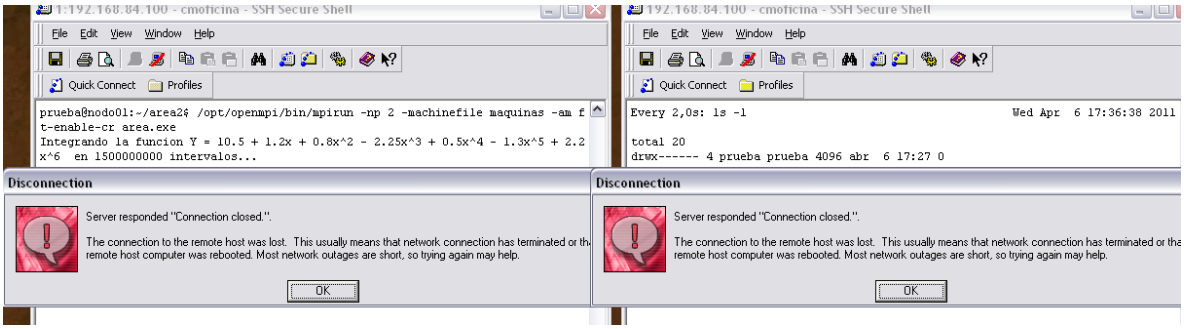


Figura 13: Suspensión de corriente

Luego de realizar la suspensión del servicio eléctrico, se realizo la restauración del proceso utilizando los checkpoints creados.

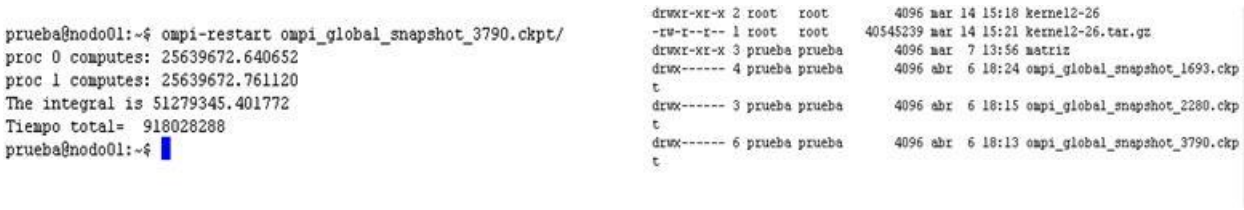


Figura 14: Reenvío prueba2

Se observó que el proceso se reinicio con éxito y finalizo sin ningún inconveniente al igual que al observar el contenido de la carpeta de usuario, se pueden encontrar los checkpoints realizados para los procesos que se reenviaron.

Después de la realización de las pruebas, se considera que la estrategia de tolerancia a fallas basada en la creación de checkpoints cumple con todos los objetivos planteados al inicio del proyecto

8. LIMITACIONES DEL PROYECTO

- La actualización de la plataforma queda limitada a las versiones del kernel que soporte el BLCR ya que si al momento de actualizar el clúster este toma una versión de kernel muy reciente y que no ha sido incluida dentro del soporte de BLCR, este no funcionara.

- La actualización de BLCR no es sencilla y debe siempre ser realizada por personal capacitado en el manejo de Computemode y en la compilación de kernel ya que una mala configuración podría dañar la plataforma completa.

- Como el proyecto está basado en BLCR, la realización de Checkpoints se limita a aplicaciones que sean soportadas en la versión de BLCR que se encuentra instalada.

9. CLONCLUCIONES

- En base al análisis y pruebas realizadas a la solución de la problemática plateada en el proyecto, se logro implementar un sistema de tolerancia a fallas el cual requirió de un análisis de diversas opciones, el estudio de las estructuras y su comportamiento, la modificación de archivos y creación de tareas programadas, con el fin de dotar a la plataforma Computemode de un sistema tolerante a fallas lo más transparente posible con el usuario y que responde a las fallas o errores planteados.

- El estudio en profundidad de computemode permitió caracterizar cada uno de los módulos de funcionamiento y contribuir con la adaptación de la plataforma a un nivel mayor que el realizado en los proyectos anteriores, como se ve en la modificación de la imagen que se distribuye a los diferentes nodos, la creación de tareas programadas para los nodos desplegados y la orientación a la transparencia de procesos a los usuarios.

- Las tablas de clasificación de errores permiten un análisis sintético de los errores en estructuras de cómputo de alto rendimiento y el manejador de colas OAR, que permite a los administradores y usuarios fácilmente identificar y clasificar los errores y decidir sobre la estrategia de solución o contingencia a realizar.

- Se establecieron los casos en que se requería el uso de puntos de restauración *checkpoints,* Proponiendo mecanismos y herramientas para la implementación de los mismos.

- Las pruebas de validación demostraron que la estrategia puede ser útil, teniendo en cuenta los resultados presentados anteriormente, garantizando el funcionamiento esperado.

- El análisis de los diferentes mecanismos de checkpoint implementados en la plataforma usada, permite conocer las características y debilidades de cada una de las opciones posibles, facilitando la implementación de la más adecuada en este proyecto, para las necesidades de los usuarios.

10. RECOMENDACIONES

- Se recomienda complementar el sistema de tolerancia a fallas mediante la integración del sistema al manejador de recursos OAR, ya que durante el desarrollo del proyecto se comprobó la integración durante el envío de trabajos, pero presento fallas en el reenvío de los mismos a través de esta herramienta, ya que OAR no cuenta con un soporte para los elementos de envío y reenvío de trabajos de BLCR.

- Se recomienda la creación de una herramienta web para los usuarios de la plataforma la cual contenga una interface grafica que facilite la ejecución de trabajos y la revisión de los datos, de manera tal que los usuarios que no conocen el lenguaje Linux puedan interactuar con mayor facilidad.

11. Bibliografía

Briatico, D., Ciuffoletti, A., & Simoncini., a. L. A distributed domino-effect free recovery algorithm. .

Buyya, R. (1999). *High Performance Cluster Computing: Architectures and Systems* (Vol. 1). Prentice Hall PTR, NJ, USA.

Cao, J., Li, Y., & Guo, M. (2005). Process Migration for MPI Applications based on Coordinated Checkpoint. *IEEE Computer society* .

Charles Fan, C. (2001). *Fault-tolerant cluster of networking elements.* Pasadena, California.

Feldgen, M. (23 de Diciembre de 2004). *Universidad de Buenos Aires Facultad de Ingenierias.* Recuperado el 16 de 11 de 2010, de Universidad de Buenos Aires Facultad de Ingenierias: http://materias.fi.uba.ar/7574/m7560t/tolerancia_fallas.PDF

H. Hargrove, P., & C. Duell, J. (2006). Berkeley lab checkpont/restart(BLCR) for Linux clusters . *46* (494).

J., A., C., A., & Cooperman, G. (2009). DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. *1* (1).

Milojicic, D. S., Dougiles, F., Paindaveine, Y., Wheeler, R., & Zhou, S. (2000). Process Migration. *32* (3).

Plank, J. S., & Li, K. (1994). Faster Checkpointing with N+1 Parity. IEEE Computer Society.

Wang, Y. (1993). *Space reclamation for uncoordinated checkpointing in message-passing systems.* Urbana-Champaign: Universidad de Illinois.

Zheng, G., Huang, C., & V. Kal, L. (2006). Performance Evaluation of Automatic Checkpoint-based fault tolerance for AMPI and Chamm++. *40* (2).

12. ANEXOS

A. Diapositivas que contienen una guía paso a paso para realizar la compilación de un kernel, estas diapositivas hacen parte de los recursos de los usuarios de la página www.SC3.uis.edu.co del grupo servicio de computación científica y de alto rendimiento de la UIS. A continuación el paso a paso para compilar un kernel de manera sencilla.

- Verifique que tiene instalado el paquete gcc, si no lo tiene instálelo, es necesario para compilar el kernel
  # aptitude install gcc.

- Obtener código fuente del kernel : El principal lugar para bajarse el código fuente entero del kernel o parches es http://www.kernel.org/.

- Descomprimir el archivo : lo convencional es descomprimir el kernel en el directorio */usr/src*, pero lo podemos realizar en cualquier directorio que deseemos y lo extraemos según el tipo de archivo descargado:
  Para quienes se descargaron el .tar.gz : **tar xvzf linux-2.6.15.tar.gz**
  Para quienes se descargaron el .tar.bz2: **tar xvjf linux-2.6.15.tar.bz2**

- Luego ingresamos al directorio
  # cd linux 2.xx.x

- Borramos archivos re generables y revertimos modificaciones que se hayan hecho anteriormente .
  # make mrproper

- Configuracion de las opciones del kernel las cuales se graban en el archivo oculto .config, nosotros usaremos el comando

make menuconfig
hay 4 tipos que podemos usar:

"make config" configura el kernel usando solamente al shell Bash (el cual es el mas común en Linux) y preguntando las opciones linea por linea.
"make menuconfig" es un método para modo texto mejorado con cuadros de dialogos a color y mayor libertad de uso, para poder usar éste método es necesario poseer los paquetes de desarrollo de programas basados en curses. # aptitude install libncurses5-dev
El método "make xconfig" es posiblemente el método mas intuitivo de todos, porque funciona bajo XWindow y es gráfico; para poder usar éste método es necesario tener instalados los paquetes de desarrollo para TCL/TK.

"make oldconfig" es un método especial, similar al "make config" pero que sirve para actualizar una anterior configuración del kernel a una nueva versión de éste, formulando solamente las preguntas nuevas. Para este no se debe ejecutar el comando "make mrproper" del principio.

- Fijamos las partes del kernel que han de ser compiladas
  #make dep

- Compilamos
  # make

- Ahora compilamos las partes del kernel que seleccionaron como modulos.
  # make modules

- Ahora compilamos las partes del kernel que seleccionaron como modulos.
  # make modules

- Creamos una imagen del kernel la cual se usa para cargar en el grub

#make bzImage
Al final nos muestra la ubicación de la imagen dentro de la carpeta del kernel

- Copiamos la bzimagen ubicada en la ruta dada al final del comando anterior, y la ubicamos en el directorio /boot
  #cp arch/ruta_indicada/boot/bzImagen /boot

Instalando el kernel

Luego de compilarlo debemos instalarlo de la siguiente manera.

Creamos el initrd para que el kernel cargue los módulos que compilamos. Realizaremos mejor la modificación de un initrd existente, en lugar de crear uno.

Para modificar un initrd, tendremos que seguir los siguientes pasos:

a) Descomprimiremos el initrd. img

b) Haremos las modificaciones en los ficheros que queramos

c) Volveremos a crear el initrd.img

a) Descomprimiremos el initrd. Img

- Ubicado en el directorio /boot
- Descomprimir el initrd
  # mkdir tmp // creamos un directorio en el que trabajaremos
  # cp initrd.img // creamos una copia del initrd que vamos a modificar
  ~/tmp # cd /tmp
  ~/tmp# mv initrd.img initrd.gz // lo pasamos de .img a .gz
  ~/tmp# gunzip initrd.gz // lo descomprimimos.

Veremos que dentro del directorio ~/tmp tendremos un fichero initrd. Para extraer los ficheros, usaremos cpio:

- ~/tmp# mkdir tmp2 //directorio para descomprimir el initrd
- ~/tmp# cd tmp2
- ~/tmp/tmp2# cpio -id < ../initrd //descomprimo en tmp2

55669 blocks

Si hacemos un ls veremos que dentro del directorio tmp2 tendremos todos los directorios y archivos del initd. Por ejemplo:

- ~/tmp/tmp2# ls
  bin conf etc init lib sbin scripts usr

Ya tenemos las carpetas a modificar entonces realizamos el paso b

b) Segundo paso: Hacer las modificaciones que necesitemos.

Ya realizados los cambios procedemos con el siguiente paso.

c) Tercer paso: Volver a crear el fichero initrd.img

Ya tenemos los ficheros debemos realizar el mismo proceso pero de forma inversa para obtener un initrd que podamos usar entonces realizamos las siguientes acciones.

- Creamos el initrd
  ~/tmp/tmp2# find . | cpio - -create - -format='newc' > ../newinitrd
  55669 blocks

- Ahora tenemos un nuevo initrd llamado newinitrd en el directorio ~/tmp. Lo comprimimos:
  ~/tmp# gzip newinitrd

- Se nos creará un fichero comprimido: newinitrd.gz. Lo renombramos:
  ~/tmp# mv newinitrd.gz initrd.img

- Luego lo copiamos en el directorio /boot
  ~/tmp# cp initrd.img /boot

Y listo. Ya tenemos nuestro initrd.img modificado.

- Por ultimo agregamos nuestro kernel al grub. Realizando modificaciones en el archivo menu.lst
  # vim /boot/gurb/menu.lst

Se recomienda imitar los que ya existen y realizar solo los cambios necesarios ejemplo:

Kernel Existente

```
title       Debian GNU/Linux, kernel 2.6.26-2-686
root        (hd0,0)
kernel      /boot/vmlinuz-2.6.26-2-686 root=/dev/hda1 ro quiet
initrd      /boot/initrd.img-2.6.26-2-686
```

Líneas que añadimos

- title **El que quieras**
  root (hd0,0)
  kernel **/boot/bzImage** root=/dev/hda1 ro quiet
  initrd **/boot/initrd.img**

- Antes de finalizar es recomendable realizar lo siguiente, ubicados dentro de la carpeta donde compilamos el kernel:
  # make modules install
  Esto te creara en la carpeta /boot los archivos:
  System.map-2.6.xx
  vmlinuz-2.6.xx

- Si desea puede cambiar en el grub el vmlinuz-2.6.xx por bzImage, el System.map-2.6.xx es necesario para poder compilar programas.

Con esto queda finalizado el proceso de creación de un kernel.

B. INSTALACION LAM/MPI

Dentro de las opciones de sistemas integrables a BLCR se encuentra LAM/MPI, su instalación y integración con BLCR se explica a continuación:

Para realizar la instalación de LAM/MPI se debe de tener en cuenta que para su compilación se necesitan las versiones 4.1 de los compiladores gcc, g++ y fortran. Esta es una delas razones por la cual no se escogió LAM/MPI,el tener que instalar una versión de compiladores 4.1 siendo que hoy en dia se encuentran en su versión 4.3, muestra claramente la des actualización que sufre la herramienta.

Teniendo instalado los paquetes de compilación se procede con los siguientes pasos.

- Descargamos la versión 7.1.4 desde la pagina del desarrollador http://www.lam-mpi.org/7.1/download.php
- Luego de descomprimir el paquete se ingresa en la carpeta obtenida, paso a seguir realizamos la configuración de la instalación.

  ./configure - -prefix=/opt/lammpi - - with-blcr=/opt/blcr - -with-rpi=crtcp -- with-cr-base-file-dir=/direccion

  Donde
  - ✓ la opción -- prefix la usamos para indicar el directorio de instalación de LAM/MPI .
  - ✓ la opción -- with-blcr se usa para indicar el directorio donde se encuentra instalado BLCR
  - ✓ la opción –with-rpi se usa para indicar el modulo RPI, si se quiere usar BLCR es modulo debe ser crtcp.
  - ✓ La opción --with-cr-base-file-dir la usamos para indicar el directorio donde que remos crear los archivos de checkpoint, esta opción se puede cambiar durante la ejecución del programa.

- Por último debe añadir las rutas de los ejecutables y librerías en el PATH y LD_LIBARY_PATH respectivamente.

Otra de las razones por la cual no se escogió esta herramienta se encuentra en su configuración ya que se debe especificar un directorio en el cual se almacenaran los checkpoints, en caso de no hacerlo el usuario deberá indicar en donde se guardaran los checkpoints, los usuarios del clúster deberán entonces añadir a su código de realización de checkpoints la ruta de almacenamiento en un archivo al cual ellos tengan acceso, caso contrario a OPENMPI que realiza la creación de checkpoints en la carpeta de usuario, dejando al usuario la posibilidad de cambiar este directorio al igual que LAM/MPI. El proceso de creación de checkpoints usando LAM/MPI es el siguiente.

Para realizar un checkpoint debe tenerse iniciado el ambiente lam por medio del comando lamboot, esto se debe hacer antes de ejecutar la aplicación.

Para ejecutar la aplicación lo realizamos mediante el comando.

$ mpirun C aplicación_mpi

Capturamos el id del proceso por medio del siguiente comando

$ ps ax -o pid,user,command | grep mpirun |grep -v grep|cut -d " " -f2

Luego de obtener el PID del proceso realizamos la petición de realización de checkpoint mediante el comando lamcheckpoint

$ lamcheckpoint PID

luego de finalizar el checkpoint del proceso se puede reiniciar en caso de presentarse una falla utilizando el archivo de contexto creado, el cual se encuentra en la carpeta que se selecciono en la instalación, el comando para restablecer un trabajo es el siguiente

$ lamrestart archivo_de_contexto

C. PARTICIPACION CLCAR 2010

Participación en la conferencia Latinoamérica de computación de alto rendimiento (CLCAR 2010), por medio de la presentación del Short Paper STRATEGIES TO FAULT TOLERANCE AND HIERARCHICAL NETWORKS INTERACTION FOR LIGHTWEIGHT CLUSTERING, formando con este parte del evento, esta participación se encuentra consignada dentro de las memorias del evento

Strategies to Fault Tolerance and hierarchical networks interaction for Lightweight Clustering

Estrategias para el manejo de Fallas e interacción entre redes jerárquicas en clústers ligeros

Mantilla Serrano, M., Orostegui Prada , S., Uribe Espinosa, R., Ruiz Sanabria, C., Escobar Ramírez, J., Barrios Hernández, C ;
Universidad Industrial de Santander
{mireyitak,86nano, rosterg85,camilo1729}@gmail.com,
carlosjaimebh@computer.org, juancaes@uis.edu.co

Abstract

Lightweight clusters are infrastructure composed by “light” resources such as desktop computers or workstations. Management and implementation of lightweight clusters is provided by different frameworks as Compute mode [1,2]. Compute mode creates a distributed computing infrastructure through the aggregation of unused computing resources.
However, it exists two critical situations: non-support to fault tolerance and not provide a management of the interaction among elements of different network levels.
In this work, we propose strategies to treat these situations, making possible the execution of longtime jobs and a management of different networks using Computemode.

Keywords: Fault tolerance; Cluster Computing; Distributed systems

Palabras claves:

Tolerancia a fallas; Computación clúster; Sistemas distribuidos.

1. INTRODUCCIÓN

Los campus universitarios cuentan con gran cantidad de equipos de cómputo interconectados, con diversos sistemas operativos predefinidos y con políticas administrativas que no permiten modificar software o el estado de los recursos. Normalmente estos se encuentran distribuidos en aulas de clase que podrían utilizarse más allá de su uso tradicional, por ejemplo, en horarios nocturnos y fines de semana siendo posible construir Clústers Ligeros (extensibles a Grid Ligeros), para soportar cálculo científico.

Existen herramientas para crear clústers ligeros por agregación de recursos en forma no intrusiva como lo es Computemode.

A pesar de las ventajas que ofrece Computemode se presenta dos limitaciones importantes: no existe un mecanismo de tolerancia a fallas que permita crear puntos de restauración (checkpoints) y en segundo lugar, esta plataforma limita los equipos de cómputo (PCs) a la subred del servidor.

De acuerdo a lo anterior y dada la naturaleza oportunista de estos tipos de infraestructura se hace necesario la creación de mecanismos de migración y checkpoints que permitan la ejecución de procesos de larga duración, así como el restablecimiento de las tareas o procesos que no llegaron a termino por fallas en la infraestructura.

En cuanto a la segunda limitación presentada, se hace oportuna la creación de una estrategia de interacción entre recursos distribuidos de una infraestructura de cómputo en redes heterogéneas.

En la siguiente sección se presenta el estado del arte. En la sección 3, se describe la propuesta general para tratar las dos problemáticas planteadas. Finalmente, presentamos en la sección 4 y 5 algunas conclusiones y los agradecimientos.

2. ESTADO DEL ARTE

La mayoría de los equipos de cómputo cuentan con una plataforma Windows como el sistema operativo residente para su uso diario. Para la implementación de clúster es común encontrar ambientes basados en distribuciones Linux, por esto su

elección para la creación de (CLUSTER y GRID).

Computemode y PelicanHPC permiten encontrar solución a la creación de estas estructuras basadas en la tecnología diskless, pero no cuentan con un manejo de tolerancia a fallas ni permiten la integración de recursos que no se encuentren dentro de la misma subred del servidor. Para las limitaciones mencionadas anteriormente existen algunos mecanismos de checkpoint y manejo de redes que trabajando en conjunto pueden ser una alternativa para mejorar el aprovechamiento de los equipos de cómputo.
Dentro de las herramientas de creación de checkpoint existen dos tipos, las que trabajan a nivel de usuario y las que lo hacen a nivel del kernel, a nivel de usuario se exige el cambio del programa y una re-compilación del mismo, un ejemplo de este tipo es Condor[20] que trabaja a este nivel.

Berkeley Lab Checkpoint/Restart (BLCR)[3], trabaja a nivel de kernel y permite a los programas que se ejecutan en Linux realizar checkpoints que pueden ser usados durante el reinicio de la máquina e incluso en otra máquina diferente.

B-OAR[5] presenta un manejo de recursos distribuidos basado en OAR[4] que es un manejador de colas desarrollado con herramientas de alto nivel, las cuales ofrecen un buen rendimiento, además de ser liviano y de fácil uso por parte del usuario. Para gestión de recursos en computación voluntaria se usa BOINC como middleware, esto con el fin de utilizar las mejores características que ofrece cada una de estas plataformas.

Otra herramienta que se centra en el manejo de recursos distribuidos basada en OAR es CiGri [6] cuyo fin es reunir los recursos no utilizados de las infraestructuras de cómputo de alto rendimiento locales para que estén disponible para un gran conjunto de tareas. Los usuarios pueden supervisar y controlar su puesto de trabajo a través de un portal web. Además el sistema proporciona mecanismos de detección de fallas, permitiendo la ejecución completa de tareas de larga duración.

3. PROPUESTA

Para la implementación de una solución se escogió Computemode que a diferencia de PelicanHPC cuenta con un manejador de colas.

[20] http://www.cs.wisc.edu/condor/overview

Se busca entonces abordar las problemáticas que se muestran a continuación.

3.1 Mecanismo de tolerancia a fallas

En el momento de implementar Computemode se encuentran las fallas señaladas en la figura 1,

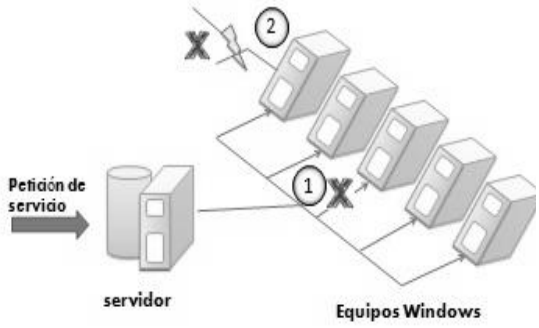


Figura1 **Fallas frecuentes infraestructura**

Numeradas por 1 y 2, se puede observar la pérdida de un nodo por falla en la infraestructura de comunicación, en la red eléctrica o por el cambio de su estado para uso de Windows; es aquí que necesitamos la implementación de un mecanismo para la creación de checkpoints. Dentro de las alternativas para la implementación de dicho mecanismo podemos encontrar Condor, el cual realiza un checkpoint a nivel de aplicación disminuyendo la sobrecarga, pero a costo de modificar el código de la aplicación. Por otro lado BLCR trabaja a nivel de kernel lo que da como resultado que se almacene completamente el estado del proceso, generando archivos de gran tamaño, lo cual ocasiona un aumento en la carga del sistema. Entre sus ventajas se encuentra permitir la creación de un mecanismo de restauración de trabajos transparente al usuario.

En el desarrollo de este trabajo se busca crear un módulo que se integre al manejador de colas, con el fin de que sea flexible y fácil de utilizar por parte del usuario. Al crear este módulo se tiene en cuenta que el usuario debe elegir el tiempo entre la creación de los checkpoints para que se acomoden al trabajo a ejecutar, ya que el tiempo de ejecución se puede ver seriamente degradado por la frecuencia de creación de estos. La integración de este mecanismo con el manejador de colas permitirá además crear una estrategia de migración de procesos, lo cual beneficiaría aplicaciones paramétricas caracterizadas por un largo tiempo de ejecución

3.2 Interacción entre redes heterogéneas

Como se observa en la figura 2, no es posible la comunicación entre nodos

de diferentes redes ya que estos se comunican a través de una red privada, que solo hace posible la comunicación de estos con su respectivo servidor. Debido a lo anterior se hace factible encontrar una forma en la cual se pueda utilizar recursos pertenecientes a diferentes servidores o subredes, un mecanismo que se podría utilizar sería el de cigri, el cual proporciona una manera trasparente de aprovechamiento de varios clústers.

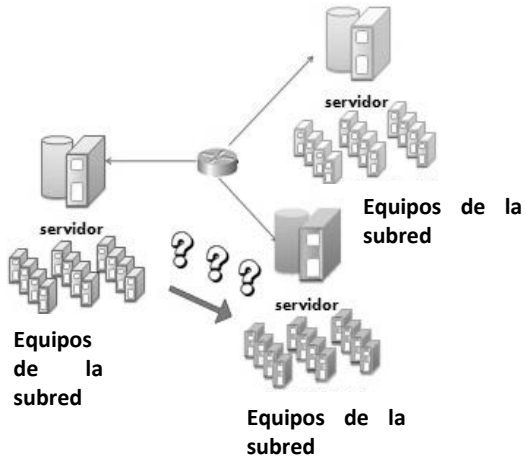


Figura2 **Arquitectura distribuida**

Cigri está diseñado como un servidor central que se encarga de recibir y enviar trabajos a los diferentes clústers registrados en el sistema, en el cual el objetivo es utilizar los instantes de tiempo en los que los recursos del clúster están libres, creando una clase de trabajos llamados best-effort los cuales cuentan con una prioridad mínima y pueden ser cancelados por otros usuario del clúster. El objetivo de la propuesta es utilizar los recursos a plenitud y evitar que estos sean cancelados como se hace en Cigri, por lo tanto se propone una modificación para que estos sean tomados como trabajos propios del clúster, implementando estas funcionalidades dentro de un conjunto de scripts que extenderán las capacidades de Computemode para formar una infraestructura distribuida.

4. CONCLUSIONES

Teniendo en cuenta la cantidad de recursos que se pueden utilizar para hacer cálculo distribuido, la creación de clúster ligeros es una alternativa altamente atractiva y se encuentra una amplia gama de plataformas que permiten la integración de recursos. Sin embargo, Computemode provee una solución integral que permite una fácil y rápida integración de recursos.
Para lograr un mecanismo adecuado de tolerancia a fallas, es necesario identificar diferentes niveles de fallas y tener en cuenta la degradación en el desempeño. Esto tiene como consecuencia que ningún mecanismo de tolerancia a fallas es absoluto. Así mismo, cualquier mecanismo de tolerancia a fallas implementado, agrega costo computacional.

El mecanismo propuesto, tiene en cuenta fallas más comunes y el costo computacional es minimizado. Igualmente, la interacción con el usuario es transparente y provee información adecuada para desarrollar nuevos mecanismos externos.

En cuanto a la interacción de recursos distribuidos en redes de diferente nivel, existen diferentes soluciones que permiten esa interacción, sin embargo ninguna ha sido implementada en Computemode.

Para aprovechar las oportunidades y ventajas de Computemode, observando las estrategias utilizadas en CiGri añadiendo la gestión de recursos distribuidos en redes de diferente nivel, es necesario tener en cuenta la prioridad de los trabajos. Por eso, nuestra propuesta busca igualar la prioridad de los trabajos.

5. AGRADECIMIENTOS

Los autores agradecen al equipo de COMPUTEMODE del LIG-Montbonnot-St Martin, Francia, dirigido por el profesor Olivier Richard.

6. REFERENCIAS

[1] Bruno Richard and Philippe Augerat, “Effective Aggregation of Idle Computing Resources for Cluster Computing”, ERICM News No. 59, October 2004.

[2] Computemode Project: http://computemode.imag.fr

[3] BLCR Frequently Asked Questions https://upc-bugs.lbl.gov//blcr/doc/html/FAQ.html#whatisblcr-Futur

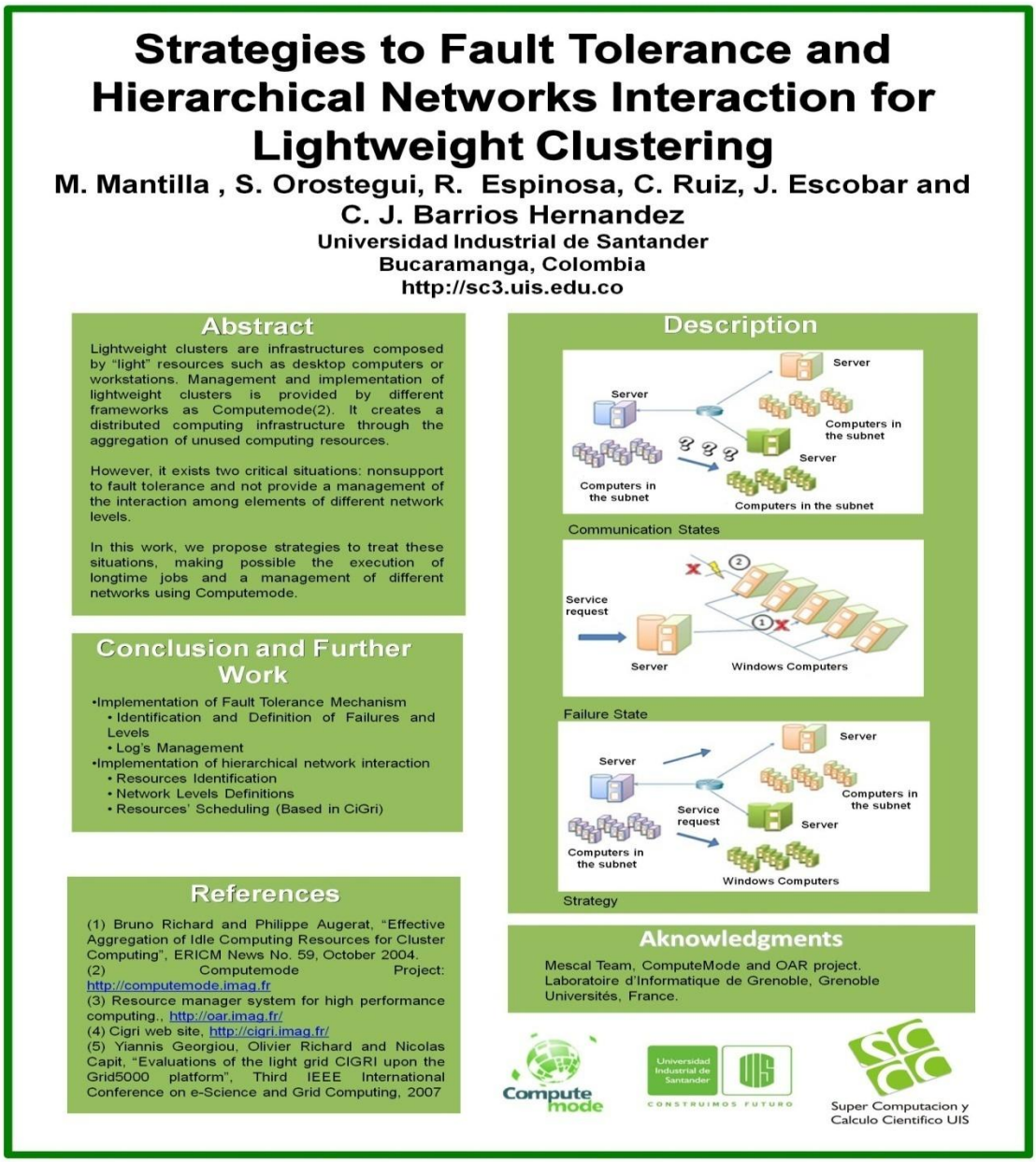
[4]Resource manager system for high performance computing., http://oar.imag.fr/

[5] *Combining OAR with the power of volunteer computing through BOINC http://oar.imag.fr/works/BOAR.html*

[6] Cigri web site, http://cigri.imag.fr/

[7]Yiannis Georgiou, Olivier Richard and Nicolas Capit, “Evaluations of the light grid CIGRI upon the Grid5000 platform”, Third IEEE International Conference on e-Science and Grid Computing, 2007

Junto con la presentación del short paper se participo en el clcar 2010 me3diante el envio del poster alusivo al contenido del short paper, este poster se expuso durante la realización de dicho evento.



Strategies to Fault Tolerance and Hierarchical Networks Interaction for Lightweight Clustering

M. Mantilla , S. Orostegui, R. Espinosa, C. Ruiz, J. Escobar and C. J. Barrios Hernandez

Universidad Industrial de Santander

Bucaramanga, Colombia

http://sc3.uis.edu.co

Abstract

Lightweight clusters are infrastructures composed by "light" resources such as desktop computers or workstations. Management and implementation of lightweight clusters is provided by different frameworks as Computemode(2). It creates a distributed computing infrastructure through the aggregation of unused computing resources.

However, it exists two critical situations: nonsupport to fault tolerance and not provide a management of the interaction among elements of different network levels.

In this work, we propose strategies to treat these situations, making possible the execution of longtime jobs and a management of different networks using Computemode.

Conclusion and Further Work

- Implementation of Fault Tolerance Mechanism
  - Identification and Definition of Failures and Levels
  - Log's Management
- Implementation of hierarchical network interaction
  - Resources Identification
  - Network Levels Definitions
  - Resources' Scheduling (Based in CiGri)

References

(1) Bruno Richard and Philippe Augerat, "Effective Aggregation of Idle Computing Resources for Cluster Computing", ERICM News No. 59, October 2004.
(2) Computemode Project: http://computemode.imag.fr
(3) Resource manager system for high performance computing., http://oar.imag.fr/
(4) Cigri web site, http://cigri.imag.fr/
(5) Yiannis Georgiou, Olivier Richard and Nicolas Capit, "Evaluations of the light grid CIGRI upon the Grid5000 platform", Third IEEE International Conference on e-Science and Grid Computing, 2007

Description

Communication States

Failure State

Strategy

Aknowledgments

Mescal Team, ComputeMode and OAR project. Laboratoire d'Informatique de Grenoble, Grenoble Universités, France.

Compute mode

Universidad Industrial de Santander

CONSTRUIMOS FUTURO

Super Computacion y Calculo Cientifico UIS