Desarrollo de un ambiente de programación paralelo para la ejecución de aplicaciones en C y JAVA

■ Maricelys Blanco Gutiérrez
email: maricelyseliza@gmail.com
Universidad Católica Andrés Bello, Caracas, Venezuela.

■ Saúl Buitrago
email: sssbuitrago@gmail.com
Universidad Católica Andrés Bello, Caracas, Venezuela.

Fecha de Recepción: 22 de mayo de 2009
Fecha de Aceptación: 19 de octubre de 2009

Resumen

El presente trabajo describe el proceso de creación de un *middleware* para un *cluster* de cómputo, denominado Cluster CSC UCAB 1, perteneciente al Centro de Súper Computación (CSC) UCAB, el cual utiliza *software* de código abierto, como Linux para el sistema operativo, y cuyo *hardware* se compone de máquinas personales de bajo costo. La finalidad está en obtener mayor transparencia ante el usuario mediante la concentración de un conjunto de herramientas que promueven el procesamiento paralelo en una sola aplicación permitiendo la ejecución de programas paralelos implementados en lenguaje C y Java que resultan ser de gran interés para otros proyectos del Centro.

Palabras claves: cluster, middleware, cómputo paralelo, mpi, torque, NFS.

1. Introducción

Los clusters se han definido como “grupos de múltiples computadoras unidas mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único computador, más potente que los comunes de escritorio”. Una de las utilidades más consideradas radica precisamente en el poder de cómputo que se puede obtener en un tiempo menor a comparación a la utilización de una sola máquina.

Desde aplicaciones de súper cómputo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, son parte de la lista de usos de los clusters.

Para lograr que ante el usuario un aglomerado de máquinas parezca una sola se debe contar con una o múltiples plataformas que logren gestionar el uso del conjunto en las tareas a ejecutar.

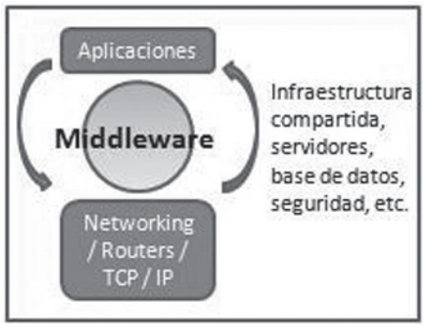

Figura 1. Middleware. [4]

Un middleware es la diferencia clave entre una red de PCs y un cluster. Es el nivel de software que se ejecuta por encima del sistema operativo y que proporciona al usuario una imagen de sistema único.

Permite un acceso uniforme a los distintos nodos de cómputo independientemente del sistema operativo que implemente cada uno de ellos. Gestiona los fallos de los nodos individuales y el equilibrio de carga de trabajo.

Entre algunos de los objetivos que debe cubrir el middleware idealmente se encuentran los siguientes:

- Jerarquía de ficheros única, ej. NFS.
- Interfaz de usuario único, ej. Entorno único.
- Espacio de entrada/ salida único, ej. Se puede tener acceso a los nodos desde los demás sin conocer la ubicación física.
- Espacio de proceso único, ej. Comunicación entre procesos como si estuvieran en la misma máquina.

En la actualidad existen diversos tipos de middleware, como por ejemplo: ParaStation, MOSIX, OpenMOSIX, Cóndor, OpenSSI, etc. Algunas de estas aplicaciones son propietarias mientras que otras son de libre distribución. Por lo general son productos de trabajos de investigación a nivel académico y científico. Cada distribución satisface las características básicas de un middleware pero el CSC UCAB requiere de uno que se ajuste a las necesidades de los proyectos que administra.

Con el propósito de aprovechar la capacidad de cómputo de un conjunto de equipos de escritorio disponibles para hacerlos funcionar como un cluster, el CSC UCAB determinó crear un middleware capaz de procesar aplicaciones paralelas implementadas en C y JAVA que utilicen las librerías MPI o PVM, haciendo uso de herramientas como TORQUE, para el procesamiento batch y balanceo de carga de los procesos, obteniendo así ejecuciones continuas de una manera sencilla de múltiples programas paralelos pertenecientes a proyectos del CSC UCAB.

2. Cluster CSC- UCAB 1

El cluster CSC- UCAB 1 es de tipo Beowulf ya que posee una arquitectura escalable de múltiples computadoras, que puede ser usada para realizar cómputo paralelo y distribuido. Este tipo de clusters son sistemas construidos con componentes de *hardware* y *software* de uso general, es decir, no contienen ningún tipo de *hardware* especializado. Los nodos están constituidos por cualquier computadora capaz de ejecutar el sistema operativo LINUX, y *software* para el desarrollo de aplicaciones paralelas.

La arquitectura del cluster consta de un nodo front-end, otro front- end auxiliar en caso de fallas y nodos trabajadores. Adicionalmente cuenta con:

Punto de red.

Switch.

Cables UTP (Unshielded Twisted Pair), con conectores RJ45.

KVM.

Tarjetas de Red (10/100Mbps), dos en los front-ends y una en los nodos trabajadores.

Emplea el protocolo TCP/IP, para el enlace de las computadoras en la LAN. El sistema operativo empleado es Linux. Utiliza el servicio SSH junto a los protocolos NIS y NFS para facilitar la administración del sistema. También maneja el estándar MPI, en las distribuciones MPICH y mpiJava para C y Java respectivamente, al igual que PVM y TORQUE.

3. Aplicaciones de Soporte para la Construcción del Middleware

PVM y MPI

El desarrollo de las principales bibliotecas de computación paralela se ha basado fundamentalmente en una *máquina virtual* o bien en el *paso de mensajes*.

Una de las bibliotecas llamada *Parallel Virtual Machine* (PVM) fue desarrollada basándose en el concepto de *máquina virtual*, esto es, una colección de recursos computacionales manejados como un computador paralelo. En cambio, otra llamada *Message Passing Interface* (MPI) se basa en el *paso de mensajes*, y aunque no tiene el concepto de máquina virtual, sí hace una abstracción de todos los recursos en términos de topología de paso de mensajes. Actualmente su uso es de mayor popularidad en comparación a PVM debido a su portabilidad y alto desempeño de comunicaciones.

En la Tabla I se destacan distintas características de amabas librerías. A continuación mencionaremos dos productos, MPICH y mpiJava, basados en la especificación MPI que fueron instalados en el cluster.

MPICH

MPICH es una de las implementaciones más robustas de MPI, habiendo evolucionado junto con el estándar, y estando disponible para una gran cantidad de plataformas. MPICH ha seguido el desarrollo de la especificación MPI y actualmente está disponible de manera portable para una gran cantidad de plataformas, entre las que se incluyen sistemas Unix comerciales (Solaris, HP UX, AIX, IRIX), máquinas masivamente paralelas (Intel Paragon, Cray) y variantes libres de Unix (Linux, BSD). También soporta arquitecturas SMP, MPP, redes de estaciones y *clusters*.

mpiJava

mpiJava es una implementación del estándar de paso de mensajes MPI para Java que requiere de la instalación previa de alguna distribución de MPI como lo es MPICH.

Comando	Descripción
mpd	Demonio que ejecuta el servicio MPICH
mpicc	Compila aplicaciones que usen librerías MPI
mpiexec	Ejecuta aplicaciones MPI compiladas
mpirun	Comando llamado internamente por mpiexec para ejecutar aplicaciones MPI
mpdboot	Comando que inicializa el servicio MPICH
mpdtrace	Muestra los hosts activos al momento

Características	PVM	MPI
Portabilidad	Media	Alta
Interoperabilidad	Alta	Media
Tolerancia a Fallos	Sí	No
Control de Procesos	Sí	MPI2: Sí
Control de Código Fuente	Dinámico	Estático
Velocidad de Comunicación	Lenta	Rápida
Comunicación Asíncrona	Sí	Sí
Topología de paso de mensajes	No	Sí
Determinar dinámicamente recursos del sistema	Sí	No

Tabla I. Características de PVM y MPI. Fuente: Adaptado de Informe # 1 Middleware de un Cluster. [1]

Comando	Descripción
mpd	Demonio que ejecuta el servicio MPICH
mpicc	Compila aplicaciones que usen librerías MPI
mpiexec	Ejecuta aplicaciones MPI compiladas
mpirun	Comando llamado internamente por mpiexec para ejecutar aplicaciones MPI
mpdboot	Comando que inicializa el servicio MPICH
mpdtrace	Muestra los hosts activos al momento

Tabla II. Algunos comandos de MPICH. [4]

Mpj es otra implementación más actualizada que funciona independiente de alguna instalación previa de MPI. Sin embargo, para efectos de este proyecto se trabajó con mpiJava pues la instalación de MPICH ya estaba contemplada.

En el ejemplo que a continuación se muestra, después de haber inicializado MPICH, el uso de mpiJava mediante la aplicación Hello.java en 2 procesos. Nótese que se trata de dos líneas con comandos específicos de java y mpiJava, que deben ser ejecutadas por cónsola en el directorio donde se encuentra la aplicación:

```
[mary@wales simple]# javac Hello.java
[mary@wales simple]# mpiexec –n 2 java Hello
received: Greetings from process 1
Hello, I am process 0
```

TORQUE

Las herramientas de balanceo de carga son gestores de recursos que proporcionan un mecanismo para encolar, controlar y ejecutar trabajos en un recurso compartido proporcionando un acceso centralizado a una serie de recursos distribuidos, de una manera inteligente, segura, justa y eficaz.

TORQUE es un gestor de recursos que proporciona control sobre trabajos batch y nodos de cómputo distribuidos. Utiliza un mecanismo de colas para ejecución de trabajos que respeta los criterios de prioridad configurados.

Algunos de los comandos que deben quedar activos después de la instalación se observan en la Tabla III.

Comando	Descripción
qsub	Envía una secuencia de comandos al servidor de colas
qstat	Muestra el estado general de trabajos
pbsnodes	Permite ver el listado de los nodos existentes
pbs_server	Inicia el servidor de trabajo
pbs_sched	Inicia el planificador de tareas
pbs_mom	Inicializa el componente de ejecutor de trabajo

Tabla III. Algunos comandos de TORQUE. [4]

Para ejecutar aplicaciones paralelas se debe usar un usuario distinto a “root” por razones de seguridad, en este ejemplo se usa el usuario “mary”:

```
[root@wales pbs]# su - mary
[mary@wales pbs]$ qsub -q cidi saludos.sh
20.wales.cidi.ve
[mary@wales pbs]# qstat -q
server: wales.CIDI.ve
Queue Memory CPU Time Walltime Node Run Que Lm State
---------- ------ -------- ---- --- --- -- -----
batch -- -- -- -- 0 1 -- E R
cidi -- 48:00:00 72:00:00 -- 0 3 -- E R
                                  ----- -----
                                  0 4
Hello World! I’m process 0 of 2 on wales.CIDI.ve
Hello World! I’m process 1 of 2 on germany.CIDI.ve
```

En el ejemplo anterior se encoló la tarea descrita en el archivo *saludos.sh*, el cual es un archivo batch con la línea de ejecución en dos nodos de la aplicación paralela en C llamada *HelloWorld.c* que utiliza las librerías MPI. Seguidamente, TORQUE lee dicho comando desde el archivo *saludos.sh* y entonces crea la tarea *20.wales.cidi.ve* (número de la tarea y nombre del nodo). Al usar el comando *qstat –q* se observa el estatus de la tarea encolada, en este caso se tienen dos nodos llamados wales.CIDI.ve y germany.CIDI.ve, dos colas activas en ejecución de tareas actuales llamadas batch y cidi, utilizándose la primera para este caso. Seguidamente se muestra la respuesta de la ejecución, la cual es configurable para almacenarla en un archivo plano.

Network File System (NFS)

NFS es un protocolo que establece el servicio de archivos distribuidos en el entorno de una red. Para

el cluster de cómputo se consideró emplearlo para el acceso de las máquinas a ficheros compartidos. De esta manera todos los nodos visualizan la misma información en los archivos configurados para ello.

Una manera de verificar la instalación y configuración del NFS en el cluster consiste en revisar en el servidor o front- end los archivos */etc/exports*, */etc/hosts. deny*, */etc/hosts.allows* y */etc/hosts*, con la finalidad de realizar las ediciones pertinentes para poder establecer la comunicación entre los nodos del cluster. Mientras que del lado del cliente basta con verificar los archivos */etc/hosts* y */etc/fstab* [9].

Para ambos casos se debe revisar que el servicio esté activo, para ello puede utilizarse desde la cónsola como usuario "root" el comando: */sbin/service nfs restart* o */sbin/service nfs start*. Para compartir una carpeta, además de incorporarla al archivo fstab del lado del cliente, se debe utilizar desde consola el comando *mount*, por ejemplo: *mount 192.168.1.1:/ home /home*, de esta manera el nodo cliente que ejecutó el comando establece el enlace con el servidor, en este caso de *IP 192.168.1.1*, pudiendo entonces compartir la carpeta */home*. Se debe destacar que el uso de estos comandos es responsabilidad de los administradores quienes los ejecutarían en caso de ser necesarios.

4. Middleware para el Cluster CSC UCAB 1

El middleware para el cluster de cómputo fue desarrollado en JAVA, permite la ejecución de aplicaciones paralelas hechas en C y Java. El objetivo principal de la plataforma es permitir que los procesos laboren de manera simultánea con distintas cantidades y tipos de datos, distribuyéndolos entre las máquinas del cluster para así disminuir la carga que tendría en un solo procesador.

Utiliza un conjunto de aplicaciones de soporte descritas en la sección anterior, engranándolas en una aplicación que llamamos middleware que facilita el trabajo al usuario, convirtiendo la transparencia en la ventaja principal de su utilización, pues elimina la tarea de la construcción de los comandos de ejecución de cada herramienta (como Torque y MPICH, por ejemplo) sino que sólo con indicar el tipo de aplicación, nombre y número de procesos deseados se puede ejecutar la aplicación paralela incluso mediante una interfaz gráfica. Adicionalmente se cuenta con la opción de facilitar una lista de aplicaciones para que sean ejecutadas en el transcurso de un tiempo determinado sin necesidad de que el usuario esté presente.

Se utilizó la arquitectura cliente- servidor en la cual existe un proceso maestro o servidor en el front- end encargado de recibir, validar, planificar los trabajos y distribuir la data facilitada por el usuario para que luego otros procesos llamados esclavos o clientes procesen la data especificada y así el servidor muestre los resultados obtenidos al usuario. Con la finalidad de definir el alcance de la aplicación se definió un modelo de dominio que puede ser apreciado en la figura 2 en el cuál se plantea a rasgos muy generales los componentes básicos del sistema de manera estática y sin detalles para poder establecer con mayor facilidad las clases que a continuación se describirán:

Entrada. Obtiene data del usuario a ser procesada, nombre e id.

Funciones. Clase que contiene las funciones básicas a ser utilizadas por la clase Entrada.

FuncionesBD. Clase que contiene las funciones relacionadas con la manipulación de la base de datos.

Insumo. Clase que define al objeto encargado de almacenar la data de entrada.

Torque. Clase que investiga estatus de TORQUE en los nodos en caso de que no se generen archivos de salidas de resultados de procesamiento batch en un tiempo predefinido.

Salida. Clase que muestra el contenido de los archivos de resultados de TORQUE.

CampoTexto. Clase de interfaz gráfica que usa métodos generales de las librerías awt y swing para garantizar portabilidad tanto de versiones de JDK como de sistemas operativos.

Contenido. Clase complementaria a CampoTexto que se encarga de la manipulación de archivos planos.

Entiéndase que del modelo de dominio explotaron conceptos más específicos para la creación de las clases arriba expuestas, por ejemplo, del dominio BasedeDatos surgió la clase FuncionesBD así como del dominio Data surge las clases Insumo y Entrada. El modelo de dominio sirve para ir rebajando el nivel de abstracción en la fase de diseño de los sistemas.

La información necesaria para que el middleware pueda trabajar se obtiene de un archivo plano de insumo llamado *data* el cual contiene la información respectiva de cada tarea. El archivo es

elaborado por el usuario y debe tener la siguiente estructura: <lenguaje><nombre><cantidad_de_proceso><naturaleza>.

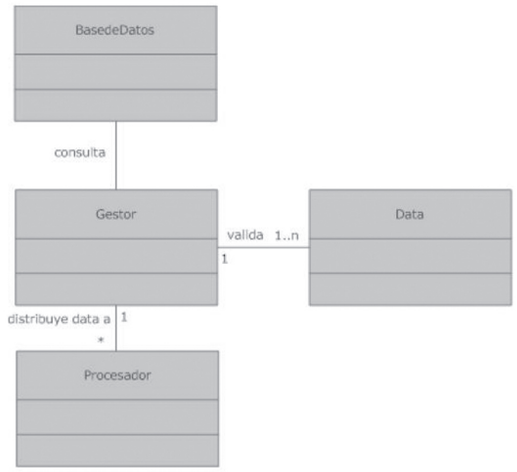

Figura 2. Modelo de Dominio. [4]

<lenguaje> indica en qué lenguaje de programación está desarrollada la aplicación, sólo se contempla C, JAVA y algunos comandos de cónsola, el <nombre> es el correspondiente al nombre del archivo principal o main de la aplicación, la <cantidad_de_procesos> indica el número de procesos requeridos para la ejecución del programa y la <naturaleza> de la aplicación determina si se trata de un programa secuencial o paralelo desarrollado en base a MPI o PVM. Este archivo puede contener una lista de varias aplicaciones, se sigue el formato descrito arriba y se separa por comas.

Para una prueba sencilla inicial se facilitó el archivo *data* como insumo. Contenido: *c problem 2 pvm, java Amen 2 mpi, c ex_in 2 secuencial*. Al terminar la ejecución del middleware se observaron los mensajes de la ejecución de cada programa así como la verificación del usuario que ejecuta la aplicación:

```
Tarea validar usuario correcta.
Lista de aplicaciones a ejecutarse (Lenguaje- Nombre-
Procesos- Tipo):
c problem 2 pvm
,
 java Amen 2 mpi
,
 c ex_in 2 secuencial

 1126.wales.cidi.ve
 1127.wales.cidi.ve
 1128.wales.cidi.ve

Procesando solicitudes, por favor esperar ...
```

Resultados. Son presentados por nombre del archivo que contiene la respuesta respectiva:

```
Nombre: Amen
Contenido:

Hello germany.CIDI.ve
received: Greetings from process 1 - wales.CIDI.ve
Nombre: ex_in.sh.out
Contenido:
tiempo 84449453
idum: -79886119
xprod 10.873127
 j = 138619568

 j = 138619568

 j = 138619568

 j = 138619568

 j = 138619568

 j = 138619568

 j = 138619568

 j = 138619568
 0.2875139  1.6477040  0.1265067
 0.1922772  1.2092212  0.5191070
 1.8914253  1.6665010  0.5758549
 0.7267339  0.3881325  0.7377759
 0.6711482  1.1695160  1.0107218
 1.3668191  0.7954742  1.0941652
 0.8154905  1.8677541  1.2604208
 1.9260110  0.6993417  1.2768101
 0.2730283  0.1733599  1.4973184
 1.6218511  0.3179289  1.8221426
Tiempo de duracion: 0.160000 milisegundos o 191740
nanosegundos

 fin calculo puntos iniciales

Nombre: problem.sh.out
Contenido:
hello, i'm a slave with tid 262146 running on machine
wales.CIDI.ve
The array of 10 elements contains 8 prime numbers.
Parallel Time: 0s 98028us
The array of 10 elemnts contains 8 prime numbers.
Sequential Time: 0s 25us
Speedup: 0.000255029
Efficiency: 0.000127515
```

Nótese que el middleware ejecuta las aplicaciones secuenciales y paralelas en mpi o pvm, en C o JAVA, que sean suministradas mediante el archivo de insumo

data. El middleware creado le permite al usuario facilitar una lista de aplicaciones sin que se preocupe por proporcionar líneas de código de ejecución para las mismas.

En relación a la interfaz, se presentaron dos tipos. Una sencilla local que consta de una ventana en la cual se puede ingresar el nombre del usuario y contraseña para luego observar los resultados de la ejecución en la misma ventana, y otra interfaz web informativa, bastante simple, con la cual se podrá observar el resultado de la última ejecución del middleware.

Esta última interfaz fue creada usando PHP y html, la cual fue probada en el laboratorio mediante la LAN, usando apache como servidor. En la interfaz local se emplea las clases awt y swing, sin intervención de paquetes propios de ningún IDE como Eclipse o Netbeans.

Para la interfaz web se creó un directorio llamado *public_html* en el directorio */home/mary*, y se habilitó la opción de apache para crear directorios de usuarios para poder acceder a esta interfaz perteneciente al usuario "mary" siempre y cuando el servicio de apache esté disponible.

Mediante cualquier nodo de la LAN perteneciente al cluster se visualiza colocando en el navegador lo siguiente: *http://200.2.13.167/~mary*, mientras que en la máquina servidor Wales se coloca *http://localhost/~mary* para garantizar el acceso. Además de la plantilla web utilizada, descargada de manera gratuita de internet, dicha interfaz está conformada por los siguientes archivos:

index.php: Contiene la página principal de la interfaz en la cual se muestra la última ejecución del middleware.

correo.php: Esta página se encarga de almacenar en un archivo plano, llamado *mensajes*, los comentarios que se hagan mediante esta interfaz para que el administrador los considere y pueda responder a los mismos. Mediante la interfaz web se puede visualizar los resultados de la última ejecución del middleware en el laboratorio del CSC UCAB, y también comunicarse con los administradores en caso de desear la inclusión de un código en particular.

Esto se debe a que los administradores, por motivos de seguridad, desean visualizar el código antes de ejecutarlo en el cluster.

Figura 3. Interfaz web Informativa.

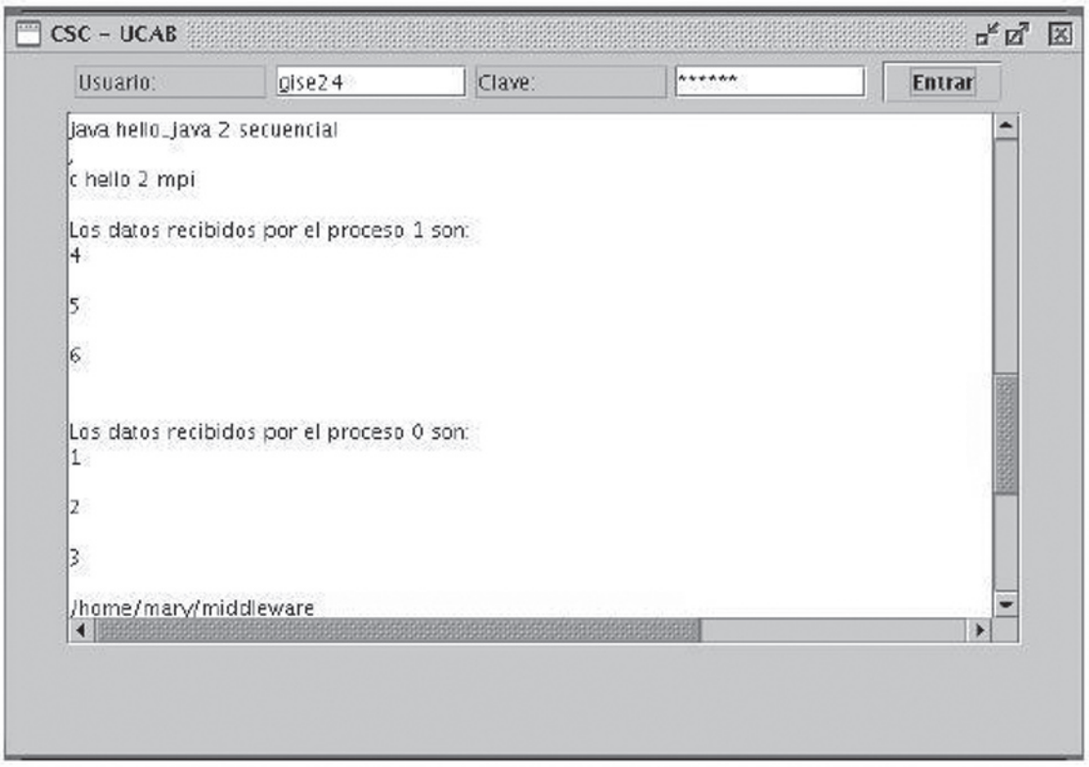

Figura 4. Interfaz local.

Es entonces posible mediante las interfaces creadas la visualización de la ejecución de tareas paralelas en C y JAVA de manera automática y continua. Basta con crear el archivo insumo *data* con las especificaciones deseadas y ejecutar la aplicación creada.

También se ejecutó una prueba de respuesta a fallas para la que se utilizó el mismo insumo para la prueba inicial de funcionamiento, con la diferencia que se eliminó uno de los hosts de la plataforma justo antes de que TORQUE encolara las tareas. El comportamiento de TORQUE fue pasivo ya que terminó la ejecución del middleware normalmente pero sin dar mensaje alguno referente a la falta del nodo.

Sin embargo al ejecutar dos veces seguidas el comando *pbsnodes*, comando de TORQUE para conocer el estatus de los nodos, apareció lo que se aprecia en la figura 5. La respuesta de dicho comando refleja que TORQUE se percató de la ausencia del nodo al mostrar la etiqueta de *status=job-exclusive* la primera vez, y después mostrar *status=down*. En este caso sólo le da tiempo de procesar la primera tarea de la cola, llamada *ls,* como se observa en la figura 7, en la cual se muestra el contenido del directorio compartido */home/mary*, en el que TORQUE tiene configurado la exposición de los resultados. Aparecen sólo los archivos de salidas preconfigurados para TORQUE *ls.sh.e456* y *ls.sh.o456*, de error y respuesta respectivamente, de la única tarea que pudo ejecutar, *ls*.

TORQUE suspende el procesamiento batch hasta que se reincorpore el nodo faltante, sin perder la lista de tareas. Efectivamente, después de que se incorpora el nodo se continúa con el procesamiento.

Ante este comportamiento se creó un módulo que consulta a TORQUE sobre su estatus para cada nodo en caso de que el sistema no encuentre archivos de salida en un tiempo predefinido.

Por ejemplo, en el caso de que algún nodo cliente no esté activo y se ejecute la aplicación, se obtiene un mensaje de error, en el que se le expresa al usuario que debe revisar el directorio llamado *salidas*, ubicado en el directorio donde se ejecuta el middleware y buscar los archivos de error generados para así conocer con exactitud la ausencia y motivo de la misma.

Al incorporarse el nodo faltante las tareas encoladas pendientes terminan su ejecución.

En caso de que el servidor falle, el administrador del sistema debe establecer a al nodo que sirve de servidor auxiliar como servidor principal.

5. Resultados

- Se seleccionó TORQUE como herramienta a instalar, configurar y utilizar para el procesamiento batch que usa el middleware por ser una de las herramientas más actualizadas, software libre y por contar con numerosa documentación.
- El middleware resultante puede procesar algunos comandos de sistema, aplicaciones tanto paralelas como secuenciales (aprovechando el procesamiento batch mediante TORQUE), desarrolladas en lenguaje C y JAVA, en el caso de las paralelas utilizando MPI y PVM.
- Se implementó una interfaz local con el uso exclusivo de las librerías awt y swing simplificando la portabilidad del middleware.

Figura 5. Respuesta ante la caída de un nodo.

6. Conclusiones

- Aunque el hardware disponible haya sido limitado, debido a la velocidad de procesamiento y cantidad de nodos, se logró implementar un middleware con diversas utilidades.
- Se utilizaron herramientas open source disponibles en Internet.
- El uso de NFS, en este caso, para la ejecución batch de aplicaciones paralelas y secuenciales resulta ser necesario para tener una sola copia común para todos los nodos, tanto de la data y aplicación a procesar como el resultado a obtener.
- El impacto de agregar al middleware otro lenguaje de programación para el procesamiento paralelo no será significativo siempre y cuando exista la librería para el procesamiento paralelo que lo soporte.

- Se configuró e implementó un servicio NFS para compartir directorios entre los nodos del cluster y un servicio apache para un usuario determinado, en la que se disfruta de una interfaz web.

```
[root@germany /]# pbsnodes
germany.CIDI.ve
state = job-exclusive
np = 1
properties = 192.168.1.3
ntype = cluster
jobs = 0/456.wales.cidi.ve
status = opsys=linux,uname=Linux germany.CIDI.ve 2.6.17-1.2142_FC4 #1 Tue J ul 11 22:41:14
EDT 2006 i686,sessions=2318 9322,nsessions=2,nusers=2,idletime=1,
totmem=581488kb,availmem=441672kb,physmem=319352kb,ncpus=1,loadave=0.12,netload=
77598047,state=free,jobs=,varattr=,rectime=962354492
wales.CIDI.ve
state = job-exclusive
np = 1
properties = 192.168.1.1
ntype = cluster
jobs = 0/456.wales.cidi.ve
status = opsys=linux,uname=Linux wales.CIDI.ve 2.6.11-1.1369_FC4 #1 Thu Jun 2 22:55:56 EDT
2005 i686,sessions=2268 11488
1268729390,nsessions=4,nusers=2,idletime=5,totmem=581700kb,availmem=385712kb,physmem=319564k
b,ncpus=1,loadave=0.20,netload=63196098,state=free,jobs=456.wales.cidi.ve,varattr=,rectime=9
62354541
[root@germany torque]# pbsnodes
germany.CIDI.ve
state = down
np = 1
properties = 192.168.1.3
ntype = cluster
status = opsys=linux,uname=Linux germany.CIDI.ve 2.6.17-1.2142_FC4 #1 Tue Jul 11 22:41:14
EDT 2006 i686,sessions=2318
9322,nsessions=2,nusers=2,idletime=1,totmem=581488kb,availmem=441672kb,physmem=319352kb,ncpu
s=1,loadave=0.12,netload=77598047,state=free,jobs=,varattr=,rectime=962354492
wales.CIDI.ve
state = free
np = 1
properties = 192.168.1.1
ntype = cluster
status = opsys=linux,uname=Linux wales.CIDI.ve 2.6.11-1.1369_FC4 #1 Thu Jun 2 22:55:56 EDT
2005 i686,sessions=2268 11488
12687,nsessions=3,nusers=2,idletime=22,totmem=581700kb,availmem=385736kb,physmem=319564kb,nc
pus=1,loadave=0.02,netload=63333230,state=free,jobs=456.wales.cidi.ve,varattr=,rectime=96235
4672
```

11. Recomendaciones

En caso de que la interfaz web informativa del middleware se publique en internet se recomienda agregar un módulo de validación de usuarios para garantizar la discreción de los resultados de las aplicaciones ejecutadas en batch.

Es recomendable que en un cluster que use un gestor de trabajos batch, la máquina que sirva de servidor sea la única conectada a Internet, los nodos no deberían estar en esa exposición, asimismo se debe revisar las colas y nodos disponibles de manera periódica.

Antes de instalar cualquier librería de paso de mensajes así como cualquier otra aplicación se debe verificar los pre-requisitos para garantizar un óptimo funcionamiento de la nueva instalación.

Para el middleware creado se recomienda automatizar la carga de los archivos a ser procesados, así como realizar un estudio para determinar las acciones necesarias para también automatizar la tolerancia a fallos, específicamente en relación a la transición de un servidor auxiliar a principal.

12. Bibliografía

[1] Uzcátegui, Mariana. Informe # 1 Middleware de un Cluster. Centro de Investigación y desarrollo de Ingeniería –CIDI- Universidad Católica Andrés Bello –UCAB-Montalbán, Caracas. [2006].

[2] Tanembaum, Andrew S. (1996). *Sistemas Operativos Distribuidos*. México: Prentice- Hall Hispanoamericana, S.A.

[3] Bosque, José Luis. *Tema 4: Middleware y Programación Paralela de Clusters*. Laboratorio de Arquitecturas Avanzadas de Computadores 5º Ing. Informática, [en línea]. Disponible en: htt//dac.escet.urjc.es/docencia/LAAC/LAAC_Tema4.pdf. [2008, 30 de enero].

[4] Blanco, M. (2008). *Desarrollo de Ambiente de Programación Paralelo para Ejecución de Aplicaciones en C y JAVA*. Trabajo de Grado, Ingeniería en Informática, Universidad Católica Andrés Bello, Caracas.

[5] Baker Mark, Carpenter Bryan, Fox Georey, Hoon Ko Sung y Lim Sang. *mpiJava: An Object- Oriented Java interface to MPI.* School of Computer Science University of Portsmouth, Southsea, Hants, UK, NPAC at Syracuse University Syracuse, New York, USA. Disponible en: http://ipdps.cc.gatech.edu/1999/java/baker.pdf. [1999].

[6] Alonso, José Miguel. *Programación de aplicaciones paralelas con MPI (Message Passing Interface)*. Facultad de Informática UPV/EHU, [en línea]. Disponible en: http://www.sc.ehu.es/acwmialj/edumat/mpi.pdf [1997, 13 de enero].

[7] Innocente, Roberto. *MPI Performance of a PC cluster at the ICTP.* Abdus Salam ICTP/Sissa, [en línea]. Disponible en: http://hpc.sissa.it/1st/index.html. [1999 , 22 de marzo].

[8] Diaz, Gilberto. *Portable Batch System*. Centro de Cálculo Científico Universidad de los Andes, [en línea]. Mérida: Venezuela. Disponible en: http://www.cecalc.ula.ve/HPCLC/slides/day_10/PBS.pdf. [2003, octubre- noviembre].

[9] Díaz, G., y Salicetti, G. (2006). *Desarrollo de plataforma de procesamiento paralelo distribuido para la ejecución de diferentes trabajos*. Trabajo de Grado, Ingeniería en Informática, Universidad Católica Andrés Bello, Caracas.

[10] Buitrago, S. 2009. *Global optimization techniques and their applications to traffic modeling*. In *Proceedings of the 9th WSEAS international Conference on Simulation, Modelling and Optimization* (Budapest, Hungary, September 03 - 05, 2009). R. Imre, M. Demiralp, and N. Mastorakis, Eds. Mathematics And Computers In Science And Engineering. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, 275-278.