BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

"ADMINISTRACIÓN Y GESTIÓN DE COLAS EN HPC"

TESIS PROFESIONAL

PARA OBTENER EL TITULO EN INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

ANGEL EDUARDO TOVAR VILLASEÑOR

ASESOR:

DR. MANUEL MARTÍN ORTIZ

PUEBLA, PUE. 2016

DEDICATORIA

A mis padres y hermano, porque todo lo que soy, se lo debo a ellos y por inculcar en mi la importancia del estudio y la superación personal.

A mis colegas por el apoyo en conocimiento

AGRADECIMIENTOS

A Dios

Por iluminar mi mente y ayudarme a tomar las decisiones correctas.

A los doctores y trabajadores del Laboratorio Nacional de Supercómputo

Por apoyarme en todo momento y darme oportunidades para continuar con mi conocimiento

A aquellas personas que siempre estuvieron conmigo, tanto amigos como familiares

Por creer en mí.

Contenido

INTRODUCCIÓN

A través de los años, desde el comienzo de la era de la computación y el uso de internet, hemos visto una asombrosa evolución de sus aplicaciones, tanto como de software a nivel hardware. Existe un área en específico de aplicaciones de cómputo que también ha tenido una evolución asombrosa, aunque actualmente, no es tan conocida: "El supercómputo". Con el incremento de las capacidades de las supercomputadoras, los científicos pueden estudiar con mayor detalle y precisión muchos fenómenos, lo cual tiene un impacto –directo o indirecto- en nuestro entendimiento del universo, y en la forma en que vivimos.

La evolución de las aplicaciones, tanto las cotidianas como las de supercómputo, se fundamenta en un incremento constante en las capacidades de las tecnologías digitales. Si el incremento de las capacidades se mantiene, las aplicaciones seguirán evolucionando, llevándonos a escenarios que hoy pueden parecer posibles solamente en la ficción.

Actualmente el "Supercómputo", tiene muchas aplicaciones, ya que recorre todo el intervalo del conocimiento humano; áreas de investigación científica como la ingeniería, física, matemáticas, medicina, geofísica, geografía, astronomía, química, ciencias de la atmosfera, ciencias nucleares, entre otras. Siendo así, el recurso más poderoso del cómputo científico y considerada como la principal herramienta del desarrollo científico y tecnológico ya que permite hacer complejas investigaciones en todas las áreas ya mencionadas.

Con esta tecnología, podemos ver de frente, la llegada de la medicina personalizada con ayuda de las ciencias genómicas y encontrando así, cura para enfermedades, que hasta ahora, estamos lejos de hallarla, podemos encontrar mejores predicciones de huracanes y tsunamis, siendo así, podemos evacuar a tiempo, poblaciones y ciudades.

A nivel industria, a través de supercómputo, se percibe una evolución destacada, procesando simulaciones y prototipos con una velocidad que anteriormente tardaba días. Así, las empresas automotrices, evitaran simular pruebas de sus prototipos a nivel físico, economizando costos y tiempos, utilizando el poder de procesamiento de equipos de supercómputo.

Debido a su alto poder de cómputo, la administración de recursos y gestión de procesos es de suma importancia, un hecho clave que permitirá al usuario explotar las actividades de investigación y al clúster mayor eficacia de cómputo, para esto, existen distintos tipos de herramientas, que gestionan desde recursos físicos hasta módulos de software, según sea requerido.

Haciendo un análisis de lo que el usuario demanda, podemos suministrar las herramientas y recursos necesarios para que el trabajo sea satisfactorio, en esta tesis se darán a conocer las herramientas más utilizadas a nivel mundial para la gestión de recursos, inicializando los conocimientos básicos, de que es y cómo funcionan los procesos, como lanzarlos, pausarlos, detenerlos, matarlos y monitorizar su estado, además de cuáles son los software libre y pagado, ejemplos de utilización en clúster pequeño y clúster grande, además de experimentación en distintos clúster.

La mayoría del software para la administración y gestión de colas en HPC es de código abierto, es decir, la codificación de la aplicación es abierta a cualquier persona, dando pauta a analizar el código, además de poder cambiar y mejorar el software, esto permite a los desarrolladores que trabajan en conjunto con los administradores de colas, poder hacer más eficaz la utilización de estas herramientas.

Conceptos Básicos.

Rendimiento: Es la efectividad del desempeño de una computadora, sobre una aplicación o un benchmark en particular.

Flops: Es una medida de la velocidad del procesamiento numérico del procesador. Son operaciones de punto flotante por segundo.

Alto Rendimiento: Gran demanda de procesamiento de datos en procesadores, memoria y otros recursos de hardware, donde la comunicación entre ellos es muy rápida.

Latencia: Tiempo de transferencia de mensajes de una interfaz a otra.

Ancho de banda: Capacidad de transferencia que tiene un canal de comunicaciones en una unidad de tiempo.

Alta disponibilidad: Disposición y acceso a los servicios al 100% de manera ininterrumpida.

Supercómputo: Cómputo masivo o comúnmente llamando HPC.

Sistema distribuido: Sistema en el que los recursos de cómputo (CPU, memoria y dispositivos I/O se comunican y trabajan entre sí mediante un sistema o tecnología de comunicación).

Programación en Paralelo: Estilo o método de programación que permite dividir en subprogramas a un programa para resolver un problema determinado.

Tecnología VIA: Protocolo de comunicación con características gran Ancho de banda y baja latencia.

Computadora Paralela: Máquina con dos o más procesadores que pueden trabajar simultáneamente y/o coordinadamente.

Memoria Compartida: En una máquina paralela existe una sola memoria que puede ser accesada por todos los procesadores.

Memoria distribuida: Cada uno de los procesadores de un multiprocesador tiene asociado a él una unidad de memoria.

Sistema de Memoria Compartida.

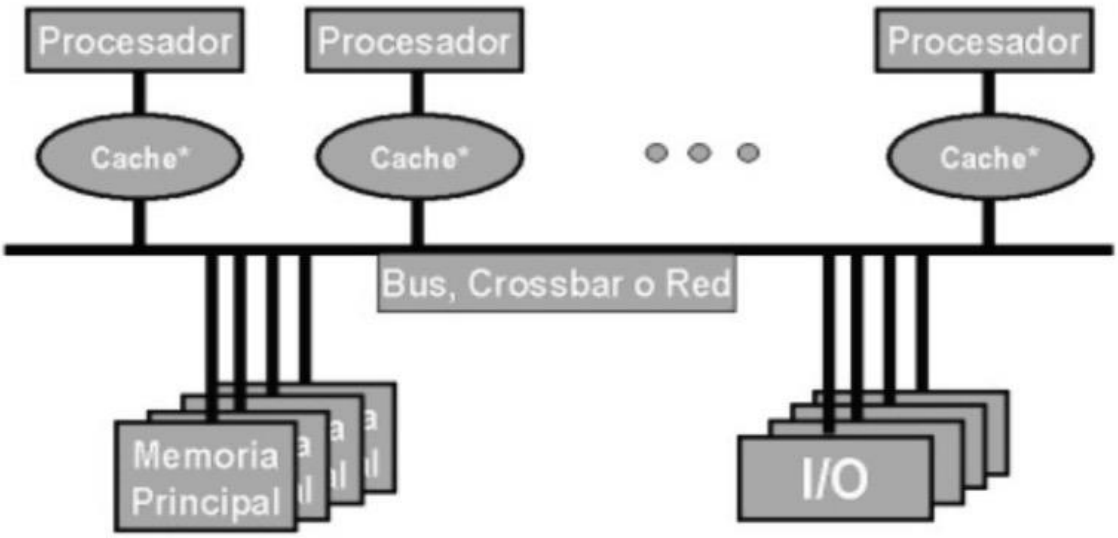


Figura 1 (“Sistema de Memoria Compartida”).

Sistema de memoria distribuida.

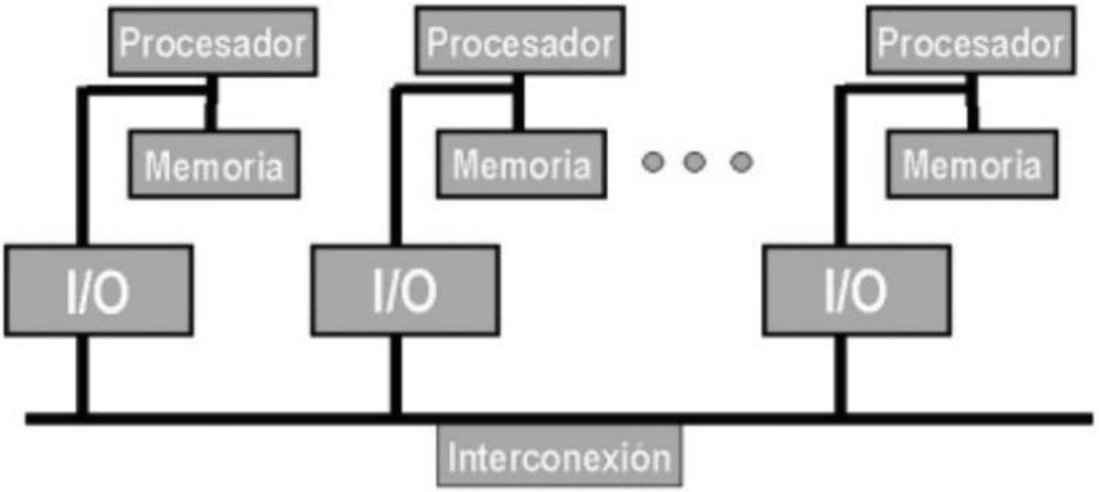


Figura 1.2 (“Sistema de memoria distribuida”)

Sistema Híbrido

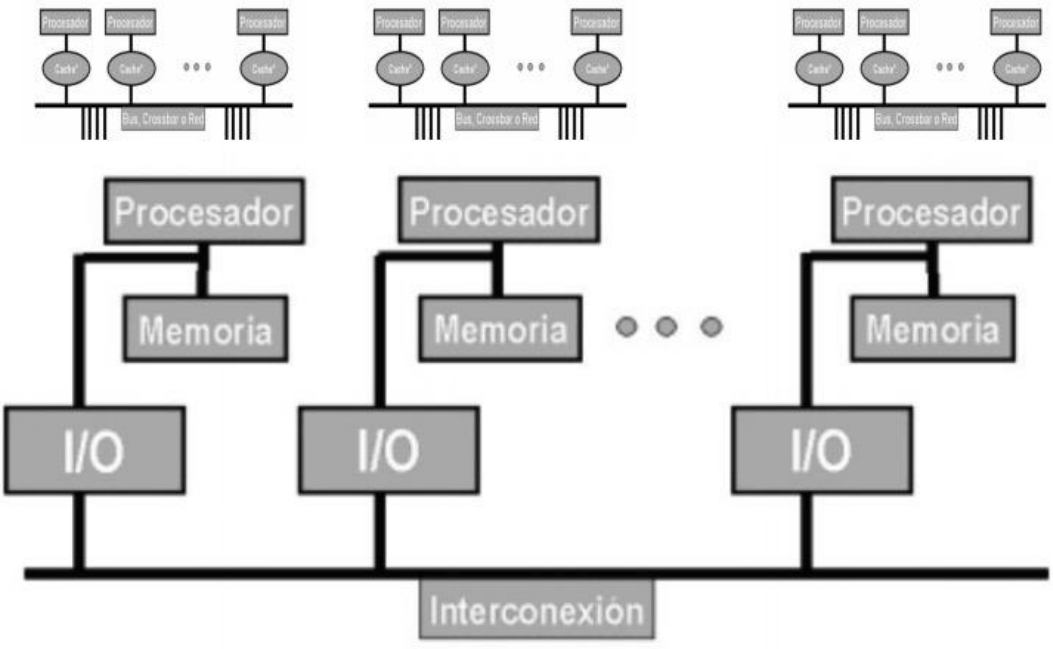


Figura 1.3 (“Sistema Híbrido”)

Con estos conceptos básicos comprenderemos más acerca de la utilización y gestión de recursos en sistemas de HPC.

CAPITULO 1: "ESTUDIO DE COLAS"

1.1. Concepto Proceso.

Un proceso es un programa en ejecución, el programa ejecutable es un conjunto de instrucciones y datos almacenados en un fichero; Cuando los componentes del programa se cargan en la memoria y lo pone en ejecución, se convierte en un proceso.

Un sistema consiste en una colección de procesos que podrían ejecutarse concurrentemente.

La entidad *"proceso"* está formada por los siguientes elementos principales.

- Imagen binaria de un programa, cargada total o parcialmente en la memoria física. La imagen binaria ésta formada por las instrucciones y datos del programa.
- Un área de memoria para almacenar datos temporales, conocida como pila.

La imagen binaria y la pila son el programa en sí mismo, pero para que el sistema operativo pueda controlar el programa hace falta una serie de estructuras de datos. Las estructuras fundamentales son.

- La tabla de páginas para traducir las direcciones virtuales generadas por el proceso en las direcciones físicas en la que se encuentra almacenado
- Una estructura de control, conocida como PCB, para que el sistema operativo pueda controlar su ejecución.

Las obligaciones del sistema operativo como gestor de procesos son:

1. Creación y eliminación de procesos
2. Planificación de procesos (procurando la ejecución de múltiples procesos maximizando la utilización del procesador).

1.2. Proceso:

Un proceso necesita recursos para realizar satisfactoriamente su tarea, entre ellos están:

1. Tiempo de CPU.

2. Memoria.
3. Archivos.
4. Dispositivos E/S.

Los recursos se asignan a un proceso:

1. Cuando se crea.
2. Durante su ejecución.

El sistema operativo se encarga de crear y borrar los procesos y de establecer comunicaciones entre ellos. La forma de gestión, de todos modos, depende del modo de trabajar de cada sistema operativo en particular.

La terminación de un proceso es la última fase de su vida, su creación y su eliminación. La terminación puede darse por diferentes razones, las cuales se manifiestan a través del estado que atraviesa el proceso al momento de ser eliminado:

Salida normal: Es cuando un proceso finaliza por voluntad del usuario, algo que ocurre constantemente durante el uso normal de un dispositivo.

Salida por error: Se trata del caso opuesto al anterior, ya que la terminación del proceso se da por la imposibilidad de continuar su ejecución.

Error fatal: Su causa es un error en el programa, algo que puede darse por diversidad razones, como ser que intente escribir en una parte de la memoria que no se encuentra accesible.

Eliminado por otro proceso: La mayoría de las razones es que se ha quedado “atrapado” en un ciclo infinito. En estos casos el único recurso para terminar el proceso parece ser la ayuda de otro, que, dependiendo del entorno, se puede ejecutar a través de la presión de un botón o de instrucciones escritas en la ventana de comandos.

1.3. Estados de un proceso.

Un proceso pasa por varios estados durante su ejecución. Los estados posibles para un proceso se muestran en la siguiente.

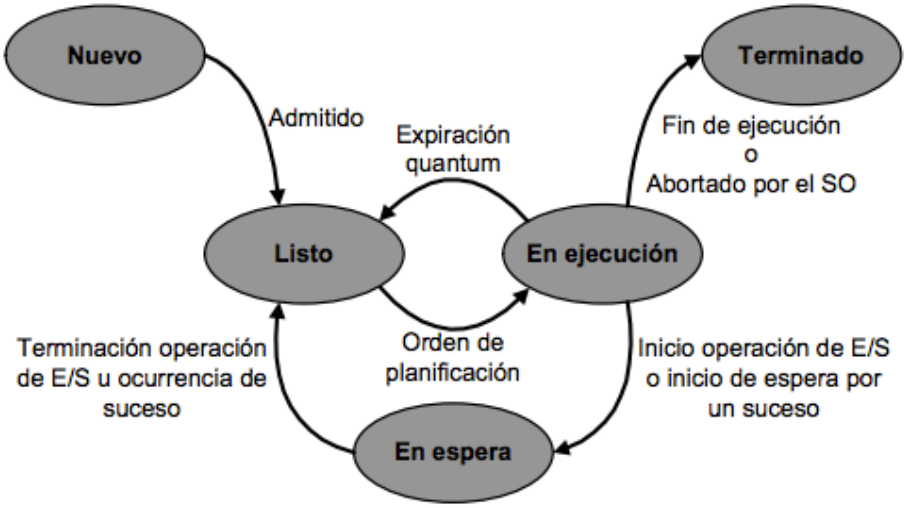


Figura 1.1. (“Estados de un proceso”)

En la anterior los nodos (nuevo, listo, etc.) representan los estados y los arcos, las acciones o eventos que llevan a un cambio de estado.

Definición de los estados:

Nuevo: El proceso se acaba de crear, pero aún no ha sido admitido en el grupo de procesos ejecutables por el sistema operativo.

Habitualmente en un sistema operativo multitarea como Mac, nada más que un proceso se crea, éste resulta admitido, pasando al estado listo. Sin embargo, esto no tiene por qué ser siempre así. Por ejemplo, en una situación de sobrecarga temporal del sistema, el SO puede decidir retardar la admisión de los procesos nuevos. Así se alivia la carga del sistema, ya que hasta que un proceso no es admitido, éste no compite por los recursos del sistema.

Listo: El proceso está esperando ser asignado al procesador para su ejecución.

En ejecución: El proceso utiliza la CPU y ésta ejecuta sus instrucciones.

En espera: El proceso está esperando a que ocurra algún suceso, como por ejemplo la terminación de una operación de E/S.

Terminado: El proceso ha sido sacado del grupo de procesos ejecutables por el sistema operativo. Después de que un proceso es marcado como terminado se liberarán los recursos utilizados por ese proceso, por ejemplo, la memoria.

1.4. Bloque de proceso.

Definición: Es una estructura de datos que permite al sistema operativo controlar diferentes aspectos de la ejecución de un proceso.

PCB = Process Control Block

Estructura típica del PCB de un proceso:

El PCB se organiza en un conjunto de campos en los que se almacena información de diversos tipos. Los campos típicamente mantenidos en el PCB de un proceso se muestran en la siguiente:

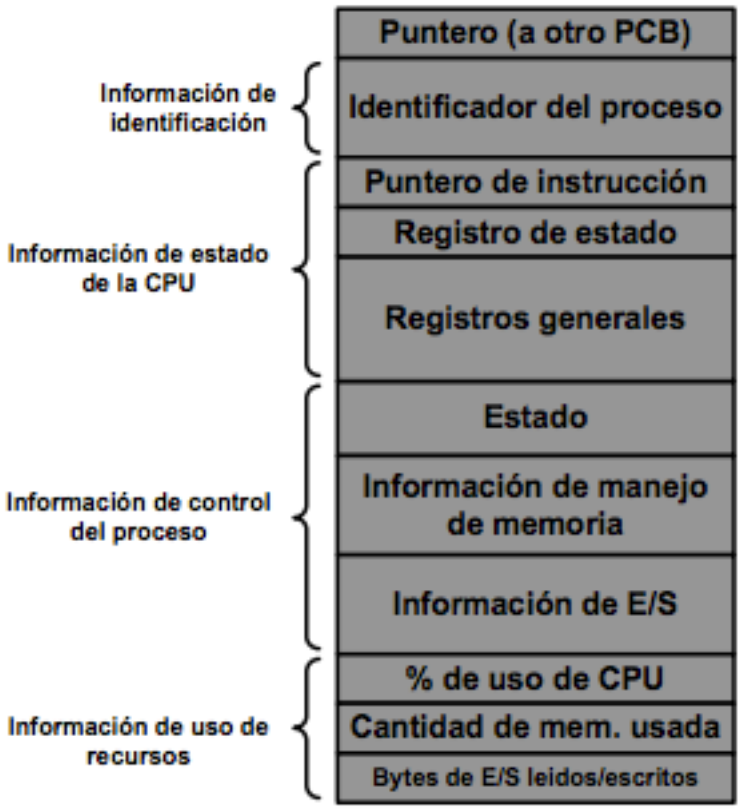


Figura 1.2 (“Estructura de un PCB”)

La información mantenida en el PCB puede clasificarse en cuatro categorías:

- **Información de identificación:**

Esta información está integrada básicamente por el identificador del proceso (PID), que es un número que identifica el proceso. Este número es diferente para todos los procesos que se encuentran en ejecución.

- **Información de estado de la CPU:**

Se trata de un conjunto de campos que almacenan el estado de los registros de la CPU cuando el proceso es suspendido

- **Información de control del proceso:**

Se trata de un conjunto de información que es utilizada por el sistema operativo para controlar diversos aspectos de funcionamiento del proceso. Pertenecen a esta categoría de información los siguientes campos:

-Estado del proceso: Listo, en ejecución, etc.
-Información de manejo de memoria: Como, por ejemplo, la dirección física de memoria en la que se ubica la tabla de páginas del proceso.
-Información de E/S: Lista de ficheros abiertos, ventanas utilizadas, etc.

- **Información del uso de recursos.**

Se trata de un conjunto de información relativa a la utilización realizada por el proceso de los recursos del sistema, por ejemplo, el porcentaje de utilización de la CPU, la cantidad de memoria usada o los bytes de E/S escritos y leídos por el proceso.

1.5. Planificación de Procesos.

El objetivo de los sistemas multitarea es mantener múltiples programas en ejecución simultáneamente, pero como la CPU sólo puede ejecutar un programa de cada vez, hay que decidir quién se ejecuta en cada momento.

Se denomina planificación (scheduling) al mecanismo utilizado por el sistema operativo para determinar qué proceso (entre los presentes en el sistema) debe ejecutarse en cada momento.

1. Planificación en sistemas de tiempo compartido

Los sistemas operativo más importantes del mercado actual se consideran de tiempo compartido.

Objetivo prioritario de estos sistemas: Garantizar que el tiempo de respuesta de los programas se mantiene en unos valores admisibles para los usuarios.

Esquema de funcionamiento: A cada proceso en ejecución se le asigna un *"quantum",* que representa el tiempo máximo que puede estar ocupando la CPU. Entonces un proceso abandona la CPU, o bien cuando se bloquea por una operación de E/S, (pasando al estado "en espera"), o bien cuando expira su *"quantum"* (pasando al estado "listo").

1.6. Colas de planificación

Son unas estructuras de datos que organizan los PCBs de los procesos que se encuentran cargados en el sistema en función de su estado.

El sistema operativo planifica los procesos en función de la información mantenida en estas colas.

Estas estructuras se forman enlazando los PCBs de los procesos mediante apuntadores.

Existen dos tipos de colas:

- **Cola de procesos listos:** Contiene a los procesos que se encuentran en el estado "listo".

  Debe indicarse una vez a más que estos procesos son los que están preparados para ser asignados a la CPU.

- **Cola de dispositivo:** Contiene los procesos que están esperando por un determinado dispositivo. Estos procesos se encuentran en el estado "En espera". Cada dispositivo tiene una cola asignada.

  Hay muchos dispositivos, como por ejemplo el disco, que son intensivamente utilizados por muchos procesos. Los procesos deben esperar ordenadamente para poder utilizar este recurso.

En la siguiente se muestra estos dos tipos de colas:

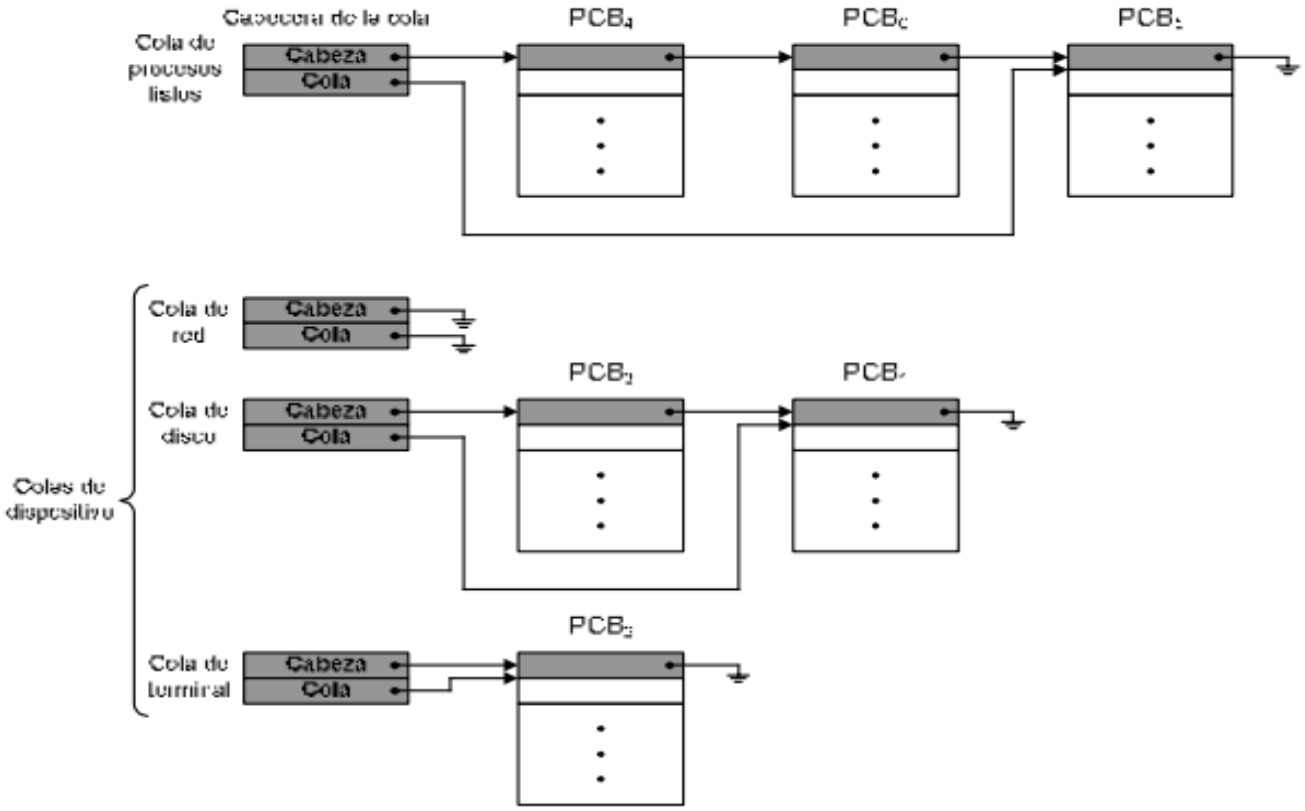


Figura 1.3 (“Tipos de colas”)

Las colas se forman enlazando mediante punteros los PCBs de los procesos. El primer campo del PCB es un apuntador que se usa para formar estas colas.

La cola tiene una cabecera que contiene dos apuntadores (llamados cabeza y cola) que se usan para apuntar al primer y último proceso de la cola.

1.7. Concepto de cambio de contexto (context switch)

Es el hecho de abandonar la ejecución de un proceso y poner en marcha otro proceso.

El cambio de contexto requiere salvar el estado que tienen los registros de la CPU justo antes de que ésta abandone el proceso que se saca de ejecución. Así después, se podrá reanudar la ejecución de este proceso, justo en el punto en el que se suspendió su ejecución. El estado de los registros de la CPU se salva en el PCB del proceso.

1.8. Concepto de swapping (Intercambio)

Se trata de un mecanismo que permite sacar procesos de ejecución, salvándolos en el disco, para luego volver a ponerlos en ejecución cuando sea requerido.

El objetivo del “swapping” es aliviar al sistema, cuando su carga de trabajo es demasiado alta, suspendiendo temporalmente en el disco unidades de trabajo (procesos). Cuando la carga del sistema baja, se ponen de nuevo en ejecución los procesos temporalmente suspendidos. Al final se conseguirá mejorar el rendimiento global del sistema multitarea.

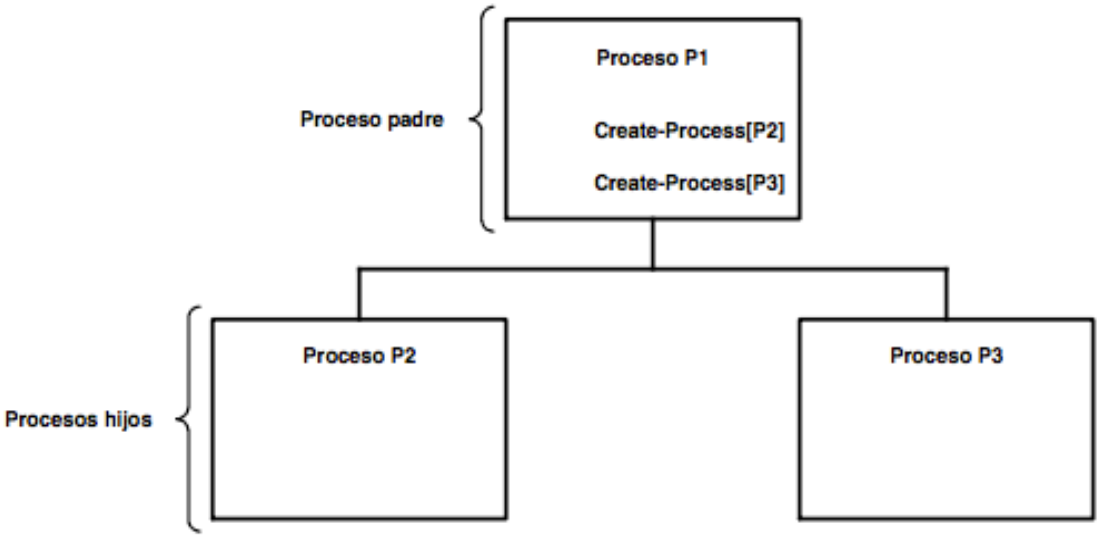
1.9. Operaciones sobre procesos.

Los procesos tienen que poder ser creados y eliminados dinámicamente en el sistema. Debido a ello, el sistema debe proporcionar facilidades para llevar a cabo estas acciones con los procesos. Las funcionalidades básicas se indican a continuación:

Creación de Procesos

Todo sistema operativo debe proporcionar un servicio Create-Process, que será utilizado por un proceso para crear otro proceso.

Al proceso que solicita el servicio Create- Process se le denomina proceso padre, y al proceso que es creado mediante este servicio, proceso hijo.



Ejemplo:

Figura 1.4 (“Proceso padre, crea proceso hijo”)

Este mecanismo de generación de procesos tiene como consecuencia que las relaciones de parentesco entre los procesos existentes en un sistema tengan estructura de árbol.

Ejemplo de árbol de procesos típico de una plataforma “unix”.

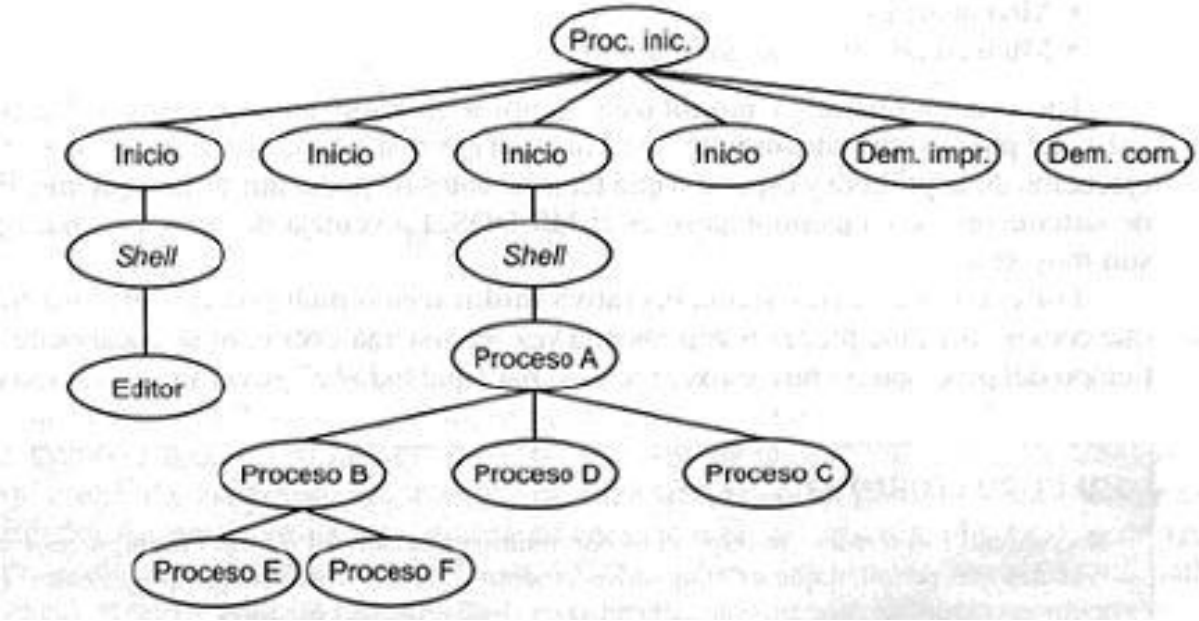


Figura 1.5 (“Ejemplo de árbol de procesos en unix”)

,

Terminación de procesos.

Un proceso puede terminar por sí mismo, o bien puede ser terminado por otro proceso, que generalmente sólo puede ser su proceso padre.

Un proceso termina por sí mismo llamando a un servicio del sistema, denominado normalmente Exit o Exit Process.

Un proceso puede terminar la ejecución de un proceso hijo llamando a un servicio del sistema, conocido normalmente como Abort o Terminate-Process.

1.10. Cooperación entre procesos.

La cooperación entre procesos requiere que estos se comuniquen. A continuación, se indican los mecanismos básicos de comunicación.

Memoria compartida. Se basa en que los procesos que desean comunicarse compartan una misma región de la memoria física. Para llevar a cabo la comunicación, uno escribe y otro lee de la región de memoria compartida.

Los procesos utilizan servicios del sistema operativo para compartir la región.

Paso de mensajes. Los procesos utilizan una pareja de servicios del sistema operativo para comunicarse. Estos servicios son conocidos habitualmente como *send y receive.*

Para llevar a cabo la comunicación un proceso ejecuta la función *send* y el otro *receive*, intercambiando de esta forma un bloque de información que recibe el nombre de mensaje.

CAPITULO 2: “COLAS Y HPC”

Actualmente, existen diferentes tipos de herramientas para la administración de colas en HPC (High Performance Computing), que permiten comprobar el estado del proceso, verificar la ejecución, recoger estadísticas de rendimiento de procesos, eliminar o pausar procesos si es necesario, entre otros;

Es importante saber cómo asignar el acceso exclusivo y/o no exclusivo de los recursos (nodos) para que los usuarios de cierta duración de tiempo puedan realizar sus procesos; seguido de proporcionar un marco de ejecución y supervisión de sus procesos que generalmente son procesos paralelos.

Gracias a estas herramientas de administración de procesos en equipos de supercómputo, podemos explotar las funcionalidades que nos ofrece.

El objetivo de estas herramientas es que los usuarios puedan especificar los recursos, obtención de respuestas rápidas, y recibir asignación segura de recursos, y para los administradores, la capacidad de comprender tanto la carga de trabajo (procesos) como los recursos disponibles. Esto incluye estados actuales, problemas y estadísticas, así como información acerca de lo que ocurre al estar procesando trabajos.

Para esto es importante mencionar el siguiente punto:

2.1.- Valor del sistema por lotes:

El sistema por lote proporciona un mecanismo para la presentación, lanzamiento y seguimiento de los procesos (trabajos) en un recurso compartido. Estos servicios respetan una de las principales responsabilidades del sistema por lote, que proporciona acceso centralizado a los recursos distribuidos. Esto simplifica en gran medida el uso de recursos distribuidos de la agrupación y permite a los usuarios una “imagen de un solo sistema” en términos de gestión de sus puestos de trabajo y los recursos informáticos agregados disponibles. Sin embargo, los sistemas de lotes deben hacer mucho más que ofrecer una visión global de la agrupación. Al igual que con muchos sistemas compartidos, complejidades surgen cuando se intenta utilizar los recursos informáticos de una manera justa y eficaz. Estas complejidades pueden conducir a malos resultados y desigualdades significativas en el uso. Con un sistema discontinuo; un planificador se le asigna a la tarea de determinar, cuándo, dónde y cómo se ejecutan los trabajos a fin de maximizar la salida del clúster. Estas decisiones se dividen en tres áreas principales:

Control de Tráfico.

Un planificador es responsable de la prevención de puestos de trabajo, de verificar la interferencia unos con otros. Si se permite que los trabajos puedan competir por los

recursos, que por lo general disminuye el rendimiento del clúster, puede retrasar la ejecución de estos mismos, y posiblemente causar problemas en uno o más lugares de trabajo. El programador es responsable de rastrear y proporcionar recursos de trabajo solicitados internamente, evitando así el uso de estos recursos por parte de otros trabajos.

Políticas.

Cuando se crean grupos u otras plataformas HPC, son creados típicamente para uno o más propósitos específicos. Estos propósitos o metas de la misión, a menudo definen diferentes reglas sobre cómo el sistema se debe utilizar y quién o qué se les permite al usarlo. Para ser eficaz, un programador debe proporcionar un conjunto de políticas que permiten que un sitio pueda asignar el comportamiento de cada sitio de trabajo y/o usuario.

Optimización.

El poder de cómputo de un clúster es un recurso limitado y con el tiempo, la demanda inevitablemente es superior a la oferta. Para esto, decisiones inteligentes de programación puede mejor significativamente la eficacia de la agrupación que resulta en: más puestos de trabajo en ejecución y más rapidez, cambios de puestos de trabajo, etc.; Sujeto a las limitaciones del control de tráfico de las políticas, que es el trabajo del programador y administrador, para usar cualquier libertad está disponible para programar trabajos de tal manera con el fin de maximizar el rendimiento del clúster.

2.2. Procesos ejecutados en distintos recursos.

Cuando nosotros realizamos cualquier código desde C# (y la mayoría de los lenguajes existentes) este algoritmo, a menos que explicitemos lo contrario, será ejecutado en forma secuencial; lo cual implica que todo el proceso será realizado sobre un mismo hilo (thread) y, por consecuencia de asignación de recursos, por un mismo micro o core. Por el contrario, si utilizamos hilos dentro de nuestro código, estamos especificando que distintas partes del código serán ejecutadas en forma concurrente haciendo que el sistema operativo asigne distintos lapsos de tiempos a cada hilo o proceso. De todas formas, estos hilos pueden ser ejecutados por el mismo procesador o no, dado que el control del proceso de cada hilo es decidido por el sistema operativo que estamos utilizando. Mediante diferentes tipos de herramientas podemos especificarle al sistema operativo que distintas porciones de nuestro código sean ejecutadas por distintos procesadores o cores.

2.3 Hilos (Threads)

Los hilos son un concepto relativamente nuevo de los SO. En este contexto, un *proceso* recibe el nombre de *proceso pesado*, mientras que un hilo recibe el nombre de *proceso ligero.* El término hilo se refiere sintáctica y semánticamente a *hilos de ejecución.*

El término multihilo hace referencia a la capacidad de un SO para mantener varios hilos de ejecución dentro del mismo proceso. Como podemos ver en la siguiente Figura (2.1).

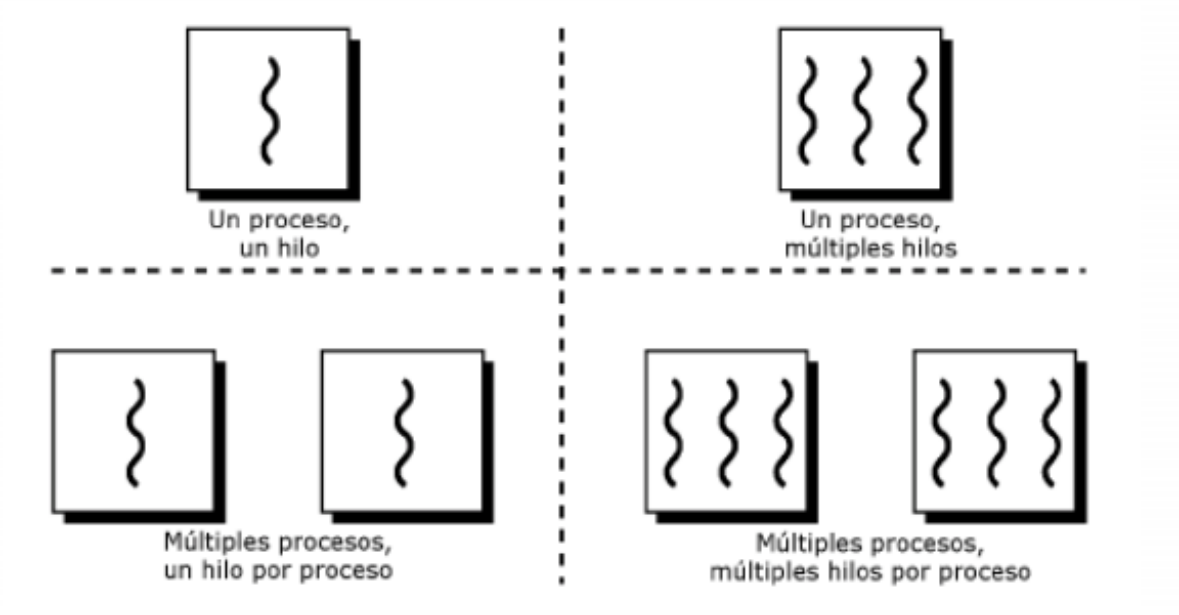


Figura 2.1 (“Hilos”)

En un SO con procesos monohilo (un solo hilo de ejecución por proceso), en el que no existe el concepto de hilo, la representación de un proceso incluye su PCB, mencionado anteriormente, un espacio de direcciones del proceso, una pila de proceso y una pila núcleo.

En un SO con procesos multihilo, sólo hay un PCB y un espacio de direcciones asociados al proceso, sin embargo, ahora hay pilas separadas para cada hilo y bloques de control para cada hilo.

2.4. Estructura de los hilos.

Un hilo (proceso ligero) es una unidad básica de utilización de la CPU, y consiste en un contador de programa, un juego de registros y un espacio de la pila.

Los hilos dentro de una misma aplicación comparten:

- La sección de código
- La sección de datos
- Los **recursos** del SO

Un proceso tradicional o pesado es igual a una tarea con un solo hilo.

Los hilos permiten la ejecución concurrente de varias secuencias de instrucciones asociadas a diferentes funciones dentro de un mismo proceso, compartiendo un mismo espacio de direcciones y las mismas estructuras de datos del núcleo.

Recursos compartidos entre los hilos

- Código
- Variables Globales
- Ficheros y dispositivos abiertos.

Recursos NO compartidos entre los hilos

- Contador del programa (cada hilo puede ejecutar una sección distinta de código)
- Registros de CPU.
- Pila para las variables locales de los procedimientos a las que se invoca después de crear un hilo.
- Estado, distintos hilos pueden estar en ejecución, listos o no bloqueados esperando un evento.

Los principales estados de un hilo son: ejecución, preparado y bloqueado y hay cuatro operaciones básicas relacionadas con el cambio de estado de los hilos:

- *Creación*: En general, cuando se crea un nuevo proceso se crea también un hilo para ese proceso. Posteriormente, ese hilo puede crear nuevos hilos dándoles un apuntador de instrucción y algunos argumentos. Ese hilo se colocará en la cola de preparados.
- *Bloqueo*: Cuando un hilo debe esperar por un suceso, se le bloquea guardando sus registros. Así el procesador pasará a ejecutar otro hilo preparado.
- *Desbloqueo*: Cuando se produce el suceso por el que un hilo se bloqueó pasa a la cola de listos.
- *Terminación*: Cuando un hilo finaliza, se liberan su contexto y sus pilas.

Nota: Un punto importante es la posibilidad de que el bloqueo de un hilo lleve al bloqueo de todo el proceso. Es decir, que el bloqueo de un hilo lleve al bloqueo de todos los hilos que lo componen, aun cuando el proceso está preparado.

En un sistema monoprocesador, la multiprogramación permite intercalar la ejecución de múltiples hilos dentro del mismo proceso:

Los hilos operan, en muchos sentidos, igual que los procesos:

- Pueden estar en uno o varios estados.
- También comparten la CPU
- Sólo hay un hilo activo (en ejecución) en un instante dado
- Un hilo dentro de un proceso se ejecuta secuencialmente
- Cada hilo tiene su propia pila y contador de programa
- Pueden crear sus propios hilos rojos

Diferencia con los procesos:

Los hilos no son independientes entre sí. Como todos los hilos pueden acceder a todas las direcciones de la tarea, un hilo puede leer la pila de cualquier otro hilo o escribir sobre ella. Aunque pueda parecer lo contrario la protección no es necesaria ya que el diseño de una tarea con múltiples hilos tiene que ser un usuario único.

Ventajas de los hilos sobre los procesos:

- Se tarda mucho menos tiempo en crear un nuevo hilo en un proceso existente que en crear un nuevo proceso.
- Se tarda mucho menos tiempo para terminar un hilo que un proceso.
- Se tarda mucho menos tiempo en conmutar entre hilos de un mismo proceso que entre procesos.
- Los hilos hacen más rápida la comunicación entre procesos, ya que, al compartir memoria y recursos, se puede comunicar entre sí sin invocar el núcleo del SO.

2.5. Hilos a nivel de núcleo

Todo el trabajo de gestión de hilos lo realiza el núcleo. En el área de la aplicación no hay código para la gestión de hilos, únicamente una Interfaz de Programas de Aplicación (API) para las funciones de gestión de hilos en el núcleo.

Este tipo de soluciones se han implementado en sistemas operativos actuales Windows y Linux. Las aplicaciones se programan como multihilo y todos los hilos de esa aplicación pertenecen al mismo proceso. El núcleo del SO mantiene la información de contexto del proceso como un todo y la de cada hilo dentro del proceso.

La principal desventada de la solución de hilos a nivel de núcleo, es que el paso de control de un hilo a otro, dentro del mismo proceso, requiere cambios de modo.

2.6. Hilos a nivel de núcleo y a nivel de usuario:

La gestión de hilos la realiza una aplicación y el núcleo no es consciente de la existencia de hilos. Se programa una aplicación como multihilo mediante una biblioteca de funciones para gestionar ULT (crear, destruir, comunicar, planificar, etc.)

Con hilos de distintos tipos, nosotros podemos mandar a ejecutar nuestros procesos a distintos nodos de cómputo para que el cálculo sea mucho más rápido y su nivel de complejidad más alto.

CAPITULO 3: “EXPLORACIÓN DE MANEJO DE COLAS EN HPC”

3.1. SLURM

Simple Linux Utility for Resource Management (SLURM) es un planificador de tareas open-source usado por muchos de las supercomputadoras a nivel mundial [1]. Provee 3 funcionalidades claves:

3.1.2. Funcionalidades clave

- Permite acceso exclusivo y/o no exclusivo a recursos (nodos) a usuarios durante un tiempo determinado de forma que puede ejecutar tareas.
- Provee un framework para iniciar, ejecutar y monitorizar tareas (típicamente tareas paralelas como MPI) a un set de nodos asignados.
- Arbitra el acceso a los recursos gestionando una cola de tareas principales.

SLURM, ha sido diseñado para manejar miles de nodos en un solo clúster, usa un algoritmo de planificación basado en la Curva de Hilbert para optimizar la localidad de las asignaciones en ordenadores paralelos.

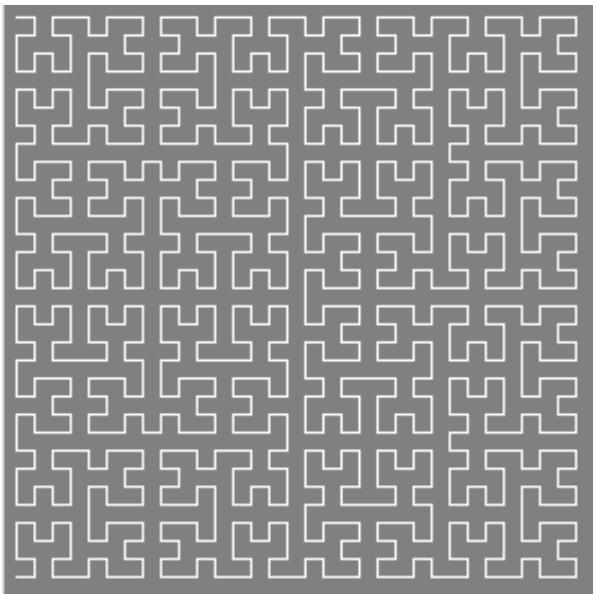


Figura 3.2 (“Quinto paso para construir la curva de Hilbert”)

3.1.3. Estructura de SLURM

El diseño de SLURM es muy modular, con docenos de plugins opcionales. En su configuración más simple, puede ser instalado y configurando en pocos minutos, mientras que configuraciones más sofisticadas proveen integración con bases de datos para cuentas de usuario, gestión de límites de recursos y priorización de carga de trabajo. SLURM trabaja así mismo con muchos meta-planificadores como Moab, Cluster Suite, Maui Cluster Scheduler y LSF.

3.1.4. Arquitectura de SLURM

Slurm consiste en un demonio slurmd que se ejecuta en cada nodo de cálculo y un demonio slurmctld que se ejecuta en un nodo de administración (con la opción de conmutación por error doble), tiene un gestor centralizado, slurmctld, para supervisar los recursos y trabajar, También puede haber un gestor de copia de seguridad para asumir esas responsabilidades en caso de fallo. Cada nodo, contiene el demonio slurmd, que puede ser comparado con una shell remoto: espera un trabajo, realiza un trabajo y devuelve el estado, continuando en espera de otro trabajo. Los demonios slurmd proporcionan comunicaciones jerárquicas tolerante a fallos. Hay un slurmdbd opcional (base de datos de slurm), que puede ser utilizado para registrar la información que es procesada por el cluster, Los comandos de usuario incluyen: sacct, salloc, sattach, sbatch, sbcast, scancel, scontrol, sinfo, smap, squeue, srun, strigger, sview. Todos los comandos se pueden ejecutar en todo el cluster.

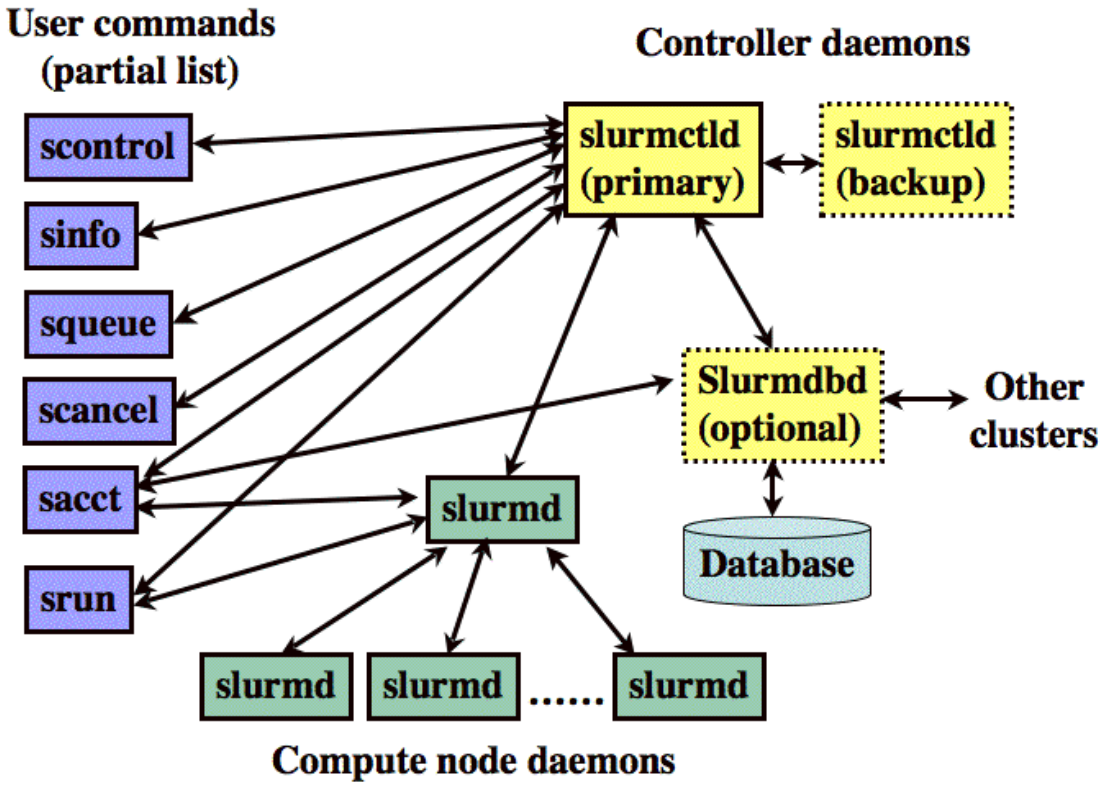


Figura 3.3 (“Componentes de slurm”)

Las entidades gestionadas por estos demonios slurm, que se muestran en la Figura 3.4, representan los nodos, el recurso de cómputo en Slurm, las particiones, terminales de trabajo, asignación de recursos a un usuario para una cantidad de tiempo especificada, conjuntos de tareas (posiblemente en paralelo) dentro de un trabajo. Las particiones pueden ser consideradas las colas de trabajo, cada una de las cuales tiene una variedad de restricciones, tales como, límite de trabajo, usuarios autorizados a utilizar, etc. Nodos dentro de una partición, hasta los recursos (nodos, procesadores, memoria, etc.) Una vez que un trabajo se le asigna a un conjunto de nodos, el usuario es capaz de iniciar un trabajo paralelo. Por ejemplo: un solo paso de trabajo puede iniciarse en todos los nodos

asignados o se pueden manejar de manera independiente (por nodo).

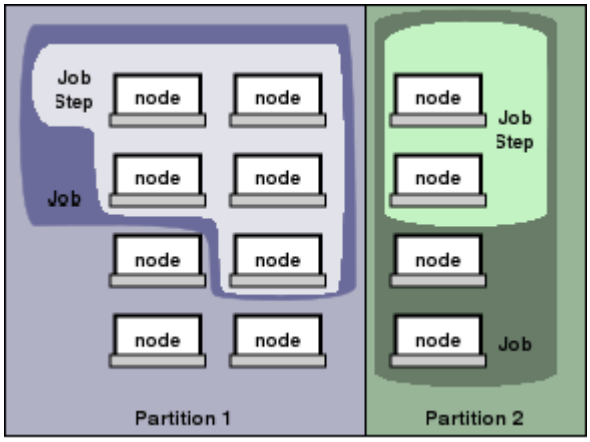


Figura 3.4 (“Entidades de slurm”)

3.1.5. Características Notables

- Altamente escalable
- Alto rendimiento
- Software Libre
- Altamente configurable con más de 100 plugins.
- Planificación compartida sobre la base de jerarquías de usuarios
- Planificación preventiva y en grupo
- Posibilidad de integración con base de datos para cuentas de usuario y configuración
- Asignación de recursos optimizada para topología de red y de nodo (sockets, cores e HyperThreading)
- Reserva avanzada
- Los nodos ociosos pueden ser apagados
- Diferentes sistemas operativos pueden ser usados para cada tarea
- Planificación para recursos genéricos (por ejemplo, unidades de procesamiento gráfico)
- Contabilidad en tiempo real a nivel de tarea (identifica tareas específicas con gran uso de CPU o memoria)

3.1.6. Plataformas Soportadas.

- AIX
- BSD – FreeBSD, NetBSD, OpenBSD
- Linux
- Mac OS X
- Solaris

3.2. Comandos básicos de SLURM.

Los usuarios que interactúan con SLURM utilizan varios comandos con los que se puede manipular las colas en esta herramienta, a continuación, se nombrarán.

SRUN – Permite correr algún proceso o hilo.

SBCAST – Transmite un archivo a todos los nodos con el proceso ejecutado.

SCANCEL – Termina un proceso en ejecución o pendiente.

SQUEUE – Monitorea la cola de trabajos.
SINFO – Monitorea las particiones configuradas en el sistema de colas y el estado general del sistema.

SACCTMGR – Para ver y modificar la información asociadas a SLURM, se utiliza con el demonio slurmdbd.

SACCT – Para visualizar datos de todos los puestos de trabajo y los pasos de trabajo en el registro contable de SLURM.

SBATCH – Presentar un script de trabajo

SALLOC - Se utiliza para asignar un trabajo a una locación (nodos).

SATTACH – Es utilizado para unir las capacidades de entrada, salida y error, manda señal a la estación de trabajo o algún trabajo en ejecución. Se puede adjuntar y separarse de estaciones de trabajo varias veces.

STRIGGER – Se utiliza para establecer, obtener o ver activadores de eventos, incluyen cosas tales como nodos que van abajo o puestos de trabajo que se acercan a su límite tiempo.

SVIEW – Es una interfaz gráfica de usuario para obtener y actualizar la información de estado de los trabajos, particiones y nodos gestionados por Slurm.

SREPORT – Es utilizado para generar reportes de los trabajos procesados en el clúster y la utilización de slurm para manejarlos, que se encuentran alojados en la base de datos de Slurm, slurmdbd.

SSTAT – Este comando muestra el status de los procesos y su información para análisis.

Nota: Todos los comandos pueden correr en cualquier parte del clúster.

3.2.1. Ejemplos de comandos de SLURM.

Estos son algunos ejemplos de comandos con el sistema de colas SLURM.

sinfo:

```
adev0: sinfo
PARTITION AVAIL  TIMELIMIT NODES  STATE NODELIST
debug*       up      30:00     2  down* adev[1-2]
debug*       up      30:00     3   idle adev[3-5]
batch        up      30:00     3  down* adev[6,13,15]
batch        up      30:00     3  alloc adev[7-8,14]
batch        up      30:00     4   idle adev[9-12]
```

Figura 3.5 (“Ejemplo de ejecución del comando ‘sinfo’”)

Como podemos ver en el ejemplo, el comando muestra la partición, si la partición está disponible, el límite de tiempo, el número de nodos, el estado y la lista de los nodos.

squeue:

```
adev0: squeue
JOBID PARTITION  NAME  USER ST  TIME NODES NODELIST(REASON)
65646     batch  chem  mike  R 24:19     2 adev[7-8]
65647     batch   bio  joan  R  0:09     1 adev14
65648     batch  math  phil PD  0:00     6 (Resources)
```

Figura 3.6 (“Ejemplo de ejecución del comando ‘squeue’”)

Como podemos ver en el ejemplo, el comando muestra los trabajos o procesos corriendo actualmente, su ID, en que partición, el nombre del proceso, el nombre del usuario, en qué estado de la cola se encuentra (pendiente, corriendo, etc.), el tiempo que lleva corriendo, los nodos utilizados, y el número de nodos o recursos.

scontrol:

```
adev0: scontrol show partition
PartitionName=debug TotalNodes=5 TotalCPUs=40 RootOnly=NO
   Default=YES Shared=FORCE:4 Priority=1 State=UP
   MaxTime=00:30:00 Hidden=NO
   MinNodes=1 MaxNodes=26 DisableRootJobs=NO AllowGroups=ALL
   Nodes=adev[1-5] NodeIndices=0-4

PartitionName=batch TotalNodes=10 TotalCPUs=80 RootOnly=NO
   Default=NO Shared=FORCE:4 Priority=1 State=UP
   MaxTime=16:00:00 Hidden=NO
   MinNodes=1 MaxNodes=26 DisableRootJobs=NO AllowGroups=ALL
   Nodes=adev[6-15] NodeIndices=5-14


adev0: scontrol show node adev1
NodeName=adev1 State=DOWN* CPUs=8 AllocCPUs=0
   RealMemory=4000 TmpDisk=0
   Sockets=2 Cores=4 Threads=1 Weight=1 Features=intel
   Reason=Not responding [slurm@06/02-14:01:24]
```

Figura 3.7 (“Ejemplo de ejecución del comando ‘scontrol’”)

Como podemos ver en el ejemplo, el comando muestra la información general de las particiones montadas.

srun:

```
adev0: srun -N3 -l /bin/hostname
0: adev3
1: adev4
2: adev5
```

Figura 3.8 (“Ejemplo de ejecución del comando ‘srun’”)

Mostrando a continuación una variante ejecutando /bin/hostname con 4 tareas, un procesador por tarea es utilizado por default (en el ejemplo no se especifica el nodo o partición donde será ejecutado.

```
adev0: srun -n4 -l /bin/hostname
0: adev3
1: adev3
2: adev3
3: adev3
```

Figura 3.9 (“Variante del comando ‘srun’”)

A continuación, se mostrará un ejemplo de cómo correr un script.

```
adev0: cat my.script
#!/bin/sh
#SBATCH --time=1
/bin/hostname
srun -l /bin/hostname
srun -l /bin/pwd

adev0: sbatch -n4 -w "adev[9-10]" -o my.stdout my.script
sbatch: Submitted batch job 469

adev0: cat my.stdout
adev9
0: adev9
1: adev9
2: adev10
3: adev10
0: /home/jette
1: /home/jette
2: /home/jette
3: /home/jette
```

Figura 3.10 (“Ejemplo de procesar un script”)

salloc:

```
tux0: salloc -N1024 bash
$ sbcast a.out /tmp/joe.a.out
Granted job allocation 471
$ srun /tmp/joe.a.out
Result is 3.14159
$ srun rm /tmp/joe.a.out
$ exit
salloc: Relinquishing job allocation 471
```

Figura 3.11 (“Ejemplo de ejecución del comando salloc”)

sbatch

```
adev0: sbatch test
srun: jobid 473 submitted

adev0: squeue
JOBID PARTITION NAME USER ST TIME  NODES NODELIST(REASON)
  473 batch     test jill R  00:00 1     adev9

adev0: scancel 473

adev0: squeue
JOBID PARTITION NAME USER ST TIME  NODES NODELIST(REASON)
```

Figura 3.12 (“Ejemplo de ejecución de ‘sbatch’”)

CAPITULO 4: “PRUEBAS DE MANEJO DE COLAS EN CLUSTER PEQUEÑO”

Como medida de seguridad, para poder conectarnos al clúster “Cuetlaxcoapan” antes debemos conectarnos a “Fénix”.

Primeramente, se inicia una conexión remota con el clúster . Existen programas que pueden hacer esto, usando el protocolo SSH: OpenSSH, puTTy, connectBox, MobaXTerm, etc.

En particular en la práctica se utilizará la terminal del sistema operativo “Mac OS X El Capitán” versión 10.11.

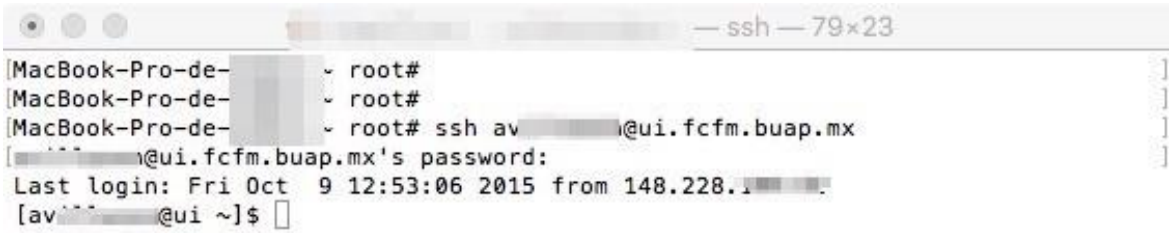
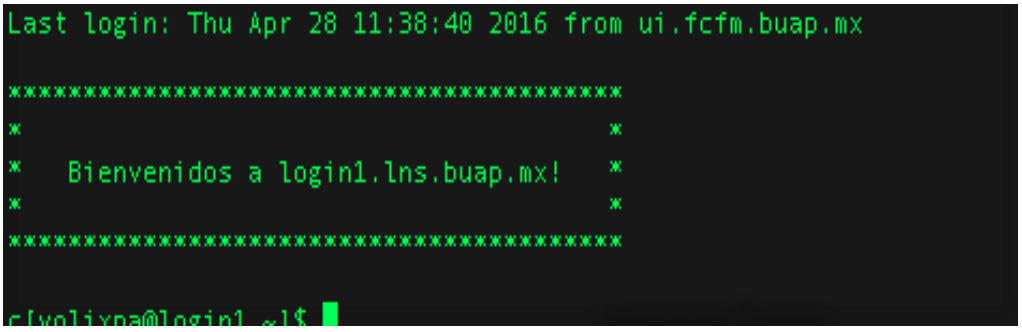


Figura 4.1 ("Conexión SSH a Fénix”)

En la Figura 4.1 podemos observar la conexión a través del protocolo SSH, al clúster Fénix. Para ejemplificar, antes de hacer una conexión a cualquier clúster se debe solicitar una cuenta con los administradores.

Una vez ya iniciada la sesión, pasamos al clúster “Cuetlaxcoapan” por el mismo protocolo.



4.2 ("Conexión SSH a Cuetlaxcoapan”)

Como podemos ver en la anterior, accedemos correctamente al clúster, aquí, podemos comenzar a hacer las pruebas con el software de gestión de colas SLURM.

4.1. Análisis del archivo de configuración SLURM.

Comenzaremos mostrando el archivo de configuración que se encuentra en el directorio /etc./slurm/slurm.conf, donde se divide la configuración en ciertas clasificaciones.

```
[yolixpa@login1 slurm]$ cat slurm.conf.example
#
# Example slurm.conf file. Please run configurator.html
# (in doc/html) to build a configuration file customized
# for your environment.
#
#
# slurm.conf file generated by configurator.html.
#
# See the slurm.conf man page for more information.
#
ClusterName=linux
ControlMachine=linux0
#ControlAddr=
#BackupController=
#BackupAddr=
#
SlurmUser=slurm
#SlurmdUser=root
SlurmctldPort=6817
SlurmdPort=6818
AuthType=auth/munge
#JobCredentialPrivateKey=
#JobCredentialPublicCertificate=
StateSaveLocation=/tmp
SlurmdSpoolDir=/tmp/slurmd
SwitchType=switch/none
MpiDefault=none
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmdPidFile=/var/run/slurmd.pid
ProctrackType=proctrack/pgid
#PluginDir=
CacheGroups=0
#FirstJobId=
ReturnToService=0
#MaxJobCount=
#PlugStackConfig=
#PropagatePrioProcess=
#PropagateResourceLimits=
#PropagateResourceLimitsExcept=
#Prolog=
#Epilog=
#SrunProlog=
#SrunEpilog=
#TaskProlog=
#TaskEpilog=
#TaskPlugin=
#TrackWCKey=no
#TreeWidth=50
#TmpFS=
#UsePAM=
#
```

Figura 4.3 ("Archivo de Configuración slurm.conf")

En la primera parte de este archivo de configuración, nosotros insertaremos información básica, nombre del clúster, en que nodo se encuentra el sistema de colas, y desde aquí, controlarlo, nombre de usuario, el puerto del sistema de colas, el tipo de autenticación que será utilizado, entre otros, algunas veces basta con Configurar estos campos. Para configuraciones más avanzadas podemos encontrar la documentación de esta configuración en la página oficial de slurm, que se encuentra en la bibliografía de esta tesis.

Otra parte importante es la configuración de la declaración de los nodos con los que cuenta en clúster, en clúster híbrido, esta configuración es esencial, debido a que a través de esta podemos definir las particiones de cada tipo de procesamiento, por ejemplo: Figura (4.4)

```
PartitionName=comp Nodes=cn0103-1,cn0103-2,cn0103-3,cn0103-4,cn
n0107-2,cn0107-3,cn0107-4,cn0108-1,cn0108-2,cn0108-3,cn0108-4,c
cn0112-2,cn0112-3,cn0112-4,cn0203-1,cn0203-2,cn0203-3,cn0203-4,
,cn0207-2,cn0207-3,cn0207-4,cn0208-1,cn0208-2,cn0208-3,cn0208-4
1,cn0212-2,cn0212-3,cn0212-4,cn0213-1,cn0213-2,cn0213-3,cn0213-
-1,cn0306-2,cn0306-3,cn0306-4,cn0307-1,cn0307-2,cn0307-3,cn0307
1-1,cn0311-2,cn0311-3,cn0311-4,cn0312-1,cn0312-2,cn0312-3,cn031
#PartitionName=terachem Nodes=tesla05-1 Default=NO MinNodes=1 M
PartitionName=tesla Nodes=tesla05-1,tesla05-2 Default=NO MinNod
#PartitionName=phi Nodes=phi05-1,phi05-2 Default=NO MinNodes=1
PartitionName=fat Nodes=fat05-1,fat05-2 Default=NO MinNodes=1 M
```

Figura 4.4 ("Ejemplos de particiones asignadas en slurm")

Como podemos ver, asigna diferentes nombres de particiones y diferentes nodos a las particiones dependiendo el tipo de procesamiento que se va a ejecutar. En esta parte de la configuración además de asignar los nodos podemos definir el mínimo de nodos, el máximo de nodos, además de ver el estado de la partición.

A continuación, mostraremos el archivo de configuración de ejemplo.

```
# Example slurm.conf file. Please run configurator.html
# (in doc/html) to build a configuration file customized
# for your environment.
#
#
# slurm.conf file generated by configurator.html.
#
# See the slurm.conf man page for more information.
#
ClusterName=linux
ControlMachine=linux0
#ControlAddr=
#BackupController=
#BackupAddr=
#
SlurmUser=slurm
#SlurmdUser=root
SlurmctldPort=6817
SlurmdPort=6818
AuthType=auth/munge
#JobCredentialPrivateKey=
#JobCredentialPublicCertificate=
StateSaveLocation=/tmp
SlurmdSpoolDir=/tmp/slurmd
SwitchType=switch/none
MpiDefault=none
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmdPidFile=/var/run/slurmd.pid
ProctrackType=proctrack/pgid
#PluginDir=
CacheGroups=0
#FirstJobId=
ReturnToService=0
#MaxJobCount=
#PlugStackConfig=
#PropagatePrioProcess=
#PropagateResourceLimits=
#PropagateResourceLimitsExcept=
#Prolog=
#Epilog=
#SrunProlog=
#SrunEpilog=
#TaskProlog=
#TaskEpilog=
#TaskPlugin=
#TrackWCKey=no
#TreeWidth=50
#TmpFS=
#UsePAM=
```

```
# TIMERS
SlurmctldTimeout=300
SlurmdTimeout=300
InactiveLimit=0
MinJobAge=300
KillWait=30
Waittime=0
#
# SCHEDULING
SchedulerType=sched/backfill
#SchedulerAuth=
#SchedulerPort=
#SchedulerRootFilter=
SelectType=select/linear
FastSchedule=1
#PriorityType=priority/multifactor
#PriorityDecayHalfLife=14-0
#PriorityUsageResetPeriod=14-0
#PriorityWeightFairshare=100000
#PriorityWeightAge=1000
#PriorityWeightPartition=10000
#PriorityWeightJobSize=1000
#PriorityMaxAge=1-0
#
# LOGGING
SlurmctldDebug=3
#SlurmctldLogFile=
SlurmdDebug=3
#SlurmdLogFile=
JobCompType=jobcomp/none
#JobCompLoc=
#
# ACCOUNTING
#JobAcctGatherType=jobacct_gather/linux
#JobAcctGatherFrequency=30
#
#AccountingStorageType=accounting_storage/slurmdbd
#AccountingStorageHost=
#AccountingStorageLoc=
#AccountingStoragePass=
#AccountingStorageUser=


# COMPUTE NODES
NodeName=linux[1-32] Procs=1 State=UNKNOWN
PartitionName=debug Nodes=linux[1-32] Default=YES MaxTime=INFINITE State=UP
```

Figura 4.5 ("Archivo de configuración SLURM.")

4.2. Formato básico de un script.

Un script es un pequeño archivo que contiene comandos para que un intérprete pueda ejecutarlos. En el caso de los scripts para SLURM, incluye tanto comandos para terminal que ya conoce como directivas para que SLURM configure la forma en que sus cálculos se ejecutarán en el clúster de supercómputo. Para hacer uno, simplemente abrimos un editor de texto, el de preferencia, y agregue comandos e instrucciones necesarias y guarde el archivo sin una extensión específica. Para ejecutarlo se utilice el comando *sbatch.*

Es sumamente importante que se mande a ejecutar sus cálculos a través de un script de SLURM, ya que sólo de ésta manera podrá hacer uso del poder de procesamiento de la supercomputadora.

Para poder correr procesos en el sistema de colas slurm, primero debemos crear un archivo denominado *job script* (sin alguna extensión de fichero específica) con los detalles de cálculo y enviarlo al sistema de colas SLURM mediante la orden:

 # sbatch job_script

donde job_script es el nombre del archivo creado.

Este archivo contendrá las instrucciones a ejecutar, desde la carga de módulos que contiene el usuario, los programas compilados a ejecutar, etc., se puede desarrollar el script desde lo más básico, hasta el más complejo.

A continuación, mostraremos el ejemplo de un script básico.

```
#!/bin/bash
#SBATCH -J test          # job name
#SBATCH -o test.o%j      # output and error file name (%j expands to
jobID)
#SBATCH -n 1             # number of MPI tasks (cores) requested
#SBATCH --ntasks-per-node=1   # task (cores) per node (maximum 24)
#SBATCH -p comp               # SLURM queue (partition)
#SBATCH -t 01:00:00           # run time (hh:mm:ss)
echo $SLURM_JOB_ID
echo $SLURM_JOB_NAME
echo $SLURM_JOB_NUM_NODES
# Load your modules here
module load compilers/intel/parallel_studio_xe_2015/15.0.1
module load tools/intel/impi/5.0.2.044

# Run your task here
mpirun -genv I_MPI_FABRICS shm:ofa YOUR_TASK
```

Figura 4.6("Ejemplo de Script Ejecutable con sbatch.")

Cómo podemos ver, en este script, imprime a pantalla, el id del proceso, su nombre y el número de nodos en los que está corriendo, solo quedaría sustituir en la última línea los nombres de los archivos de entrada (*input*) y salida (*output*) de su cálculo, y de ser necesario, el número de *cores* y tiempo de ejecución requeridos.

A continuación, ejecutaremos un script de ejemplo en el clúster “Cuetlaxcopan” que fue editado con el comando “*vim*” llamado prueba.sh

```
[yolixpa@login1 test]$ ls
prueba  prueba.sh
[yolixpa@login1 test]$ sbatch prueba.sh
Submitted batch job 170753
[yolixpa@login1 test]$ ls
prueba  prueba.sh  test.o170753
[yolixpa@login1 test]$ cat test.o170753
170753
test
```

Figura 4.7 ("Ejecución de script”)

En este ejemplo, podemos ver que se encuentra el script, y lo ejecutamos con el comando sbatch, como resultado, arroja el ID que ha sido asignado por SLURM, además de agregar el archivo creado por el mismo script, después se imprime en pantalla el archivo creado, y obtenemos el ID del proceso y el nombre que definimos en el script.

Todos los scripts ejecutados nos dan un archivo de salida.

Este es el ejemplo básico de crear, ejecutar y analizar los resultados de un script en el clúster.

4.3. Concepto MPI.

"Message Passing Interface", Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

El paso de mensajes es una técnica empleada en programación concurrente para aportar sincronización entre procesos y permitir la exclusión mutua, de manera similar a como se hace con los semáforos, monitores, etc.

Su principal característica es que no precisa de memoria compartida, por lo que es muy importante en la programación de sistemas distribuidos.

Los elementos principales que intervienen en el paso de mensajes son el proceso que envía, el que recibe y el mensaje.

Dependiendo de si el proceso que envía el mensaje espera a que el mensaje sea recibido, se puede hablar de paso de mensajes síncrono o asíncrono. En el paso de mensajes asíncrono, el proceso que envía, no espera a que el mensaje sea recibido, y continúa su ejecución, siendo posible que vuelva a generar un nuevo mensaje y a enviarlo antes de que se haya recibido el anterior. Por este motivo se suelen emplear buzones, en los que se almacenan los mensajes a espera que un proceso los reciba. Generalmente empleando este sistema, el proceso que envía mensajes sólo se bloquea o para, cuando finaliza su ejecución, o si el buzón está lleno. En el paso de mensajes síncrono, el proceso que envía el mensaje espera a que un proceso lo reciba para continuar su ejecución. Por esto se suele llamar a esta técnica encuentro, o rendezvous. Dentro del paso de mensajes síncrono se engloba a la llamada a procedimiento remoto, muy popular en las arquitecturas cliente/servidor.

La Interfaz de Paso de Mensajes (conocido ampliamente como MPI, siglas en inglés de Message Passing Interface) es un protocolo de comunicación entre computadoras. Es el estándar para la comunicación entre los nodos que ejecutan un programa en un sistema de memoria distribuida. Las implementaciones en MPI consisten en un conjunto de bibliotecas de rutinas que pueden ser utilizadas en programas escritos en los lenguajes de programación C, C++, Fortran y Ada. La ventaja de MPI sobre otras bibliotecas de paso de mensajes, es que los programas que utilizan la biblioteca son portables (dado que MPI ha sido implementado para casi toda arquitectura de memoria distribuida), y

rápidos, (porque cada implementación de la biblioteca ha sido optimizada para el hardware en la cual se ejecuta).

4.4. Características MPI

- Estandarización.
- Portabilidad: multiprocesadores, multicomputadores, redes, heterogéneos, ...
- Buenas prestaciones.
- Amplia funcionalidad.
- Existencia de implementaciones libres (mpich, LAM-MPI, ...)

La especificación detalla las funciones que se pueden utilizar, no el modo como se compilan y lanzan-ejecutan los programas, lo cual puede variar de una implementación a otra.

Siguiendo el modelo SPMD, el usuario escribirá su aplicación como un proceso secuencial del que se lanzarán varias instancias que cooperan entre sí.

Los procesos invocan diferentes funciones MPI que permiten

- iniciar, gestionar y finalizar procesos MPI
- comunicar datos entre dos procesos
- realizar operaciones de comunicación entre grupos de procesos
- crear tipos arbitrarios de datos

4.5. Diferencia entre MPI y OpenMPI

OpenMPI, es una implementación de la interfaz de paso de mensajes MPI. OpenMPI se caracteriza por su alta eficiencia y prestaciones para la ejecuciòn en entornos distribuidos.

MPI y OpenMPI, tienen sus propias ventajas y limitaciones. OpenMPI, es relativamente fácil de implementar y requiere muy pocas directivas pragma para lograr las tareas deseadas. OpenMPI, se puede utilizar en función recursiva, por lo que podría recorrer un árbol binario. Sin embargo, sufre de un problema de limitación de memoria para los cálculos que requieren mucha memoria.

MPI normalmente, trabaja con cálculos de gran memoria, además de poder utilizar memoria compartida.

Por lo que, OpenMPI, es utilizado para procesos dirigidos, y MPI, es utilizado para procesos paralelos

4.6. Ejecución de programas paralelos con MPI y SLURM.

Las ejecuciones de programas paralelos con MPI y SLURM, pueden ser muy poderosos, ya que definiendo los parámetros correctos podríamos hacer uso de un clúster completo, si la tarea, así lo requiere, además de otros módulos de programación paralela, como CUDA.

A continuación, haremos un ejemplo de un programa básico desarrollado con MPI, el cual mostrará los procesos que nosotros indiquemos a través de los parámetros de ejecución.

Nuestro programa será llamado: prueba_mpi_p.c

```
#include<stdio.h>
#include<mpi.h>

main(int argc, char **argv)
{

  int ierr, num_procs, my_id;

  ierr = MPI_Init(&argc, &argv);

  ierr = MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
  ierr = MPI_Comm_size(MPI_COMM_WORLD, &num_procs);

  printf("Hola mundo!, yo soy el proceso %i de los %i procesos totales\n", my_id, num_procs);

  ierr = MPI_Finalize();



}
```

Figura 4.8 (“Ejemplo programa en MPI”)

Como podemos ver, utilizamos las cabeceras básicas de programación.

A continuación, generamos un script, en donde utilizaremos, las directivas de SLURM, compilaremos y ejecutaremos, asignando un archivo de salida para el resultado del programa, y SLURM va a generar un archivo de salida para log.

```
#!/bin/bash
#SBATCH -J test          # job name
#SBATCH -o test.o%j      # output and error file name (%j expands to jobID)
#SBATCH -N 1             # number of nodes
#SBATCH --ntasks-per-node=24  # task (cores) per node (maximun 24)
#SBATCH -p comp              # SLURM queue (partition)
#SBATCH -t 01:00:00          # run time (hh:mm:ss)
echo $SLURM_JOB_ID
echo $SLURM_JOB_NAME
echo $SLURM_NUM_NODES


#Load your modules here
module load compilers/intel/parallel_studio_xe_2015/15.0.1
module load tools/intel/impi/5.0.2.044

#Run your task here
mpicc prueba_mpi_p.c -o mpi_salida
mpirun -np 24  mpi_salida  > sal.txt
```

Figura 4.9 (“Script con compilación MPI”)

En este script, definimos las directivas de SLURM, cargamos los módulos correspondientes para la ejecución de tareas paralelas, compilamos nuestro programa y ejecutamos especificando que este proceso se repetirá 24 veces, en un nodo del clúster, del cual, ocupará 24 cores, por lo que ejecutará 1 tarea por core.

Ya ejecutado con el comando SBATCH, SLURM, nos arrojará el ID del JOB, también generando el archivo de salida, que nosotros asignamos en el programa, un archivo con extensión .o (Archivo log, que nos dirá si ocurrió algún problema durante la ejecución.) mostrando como resultado:

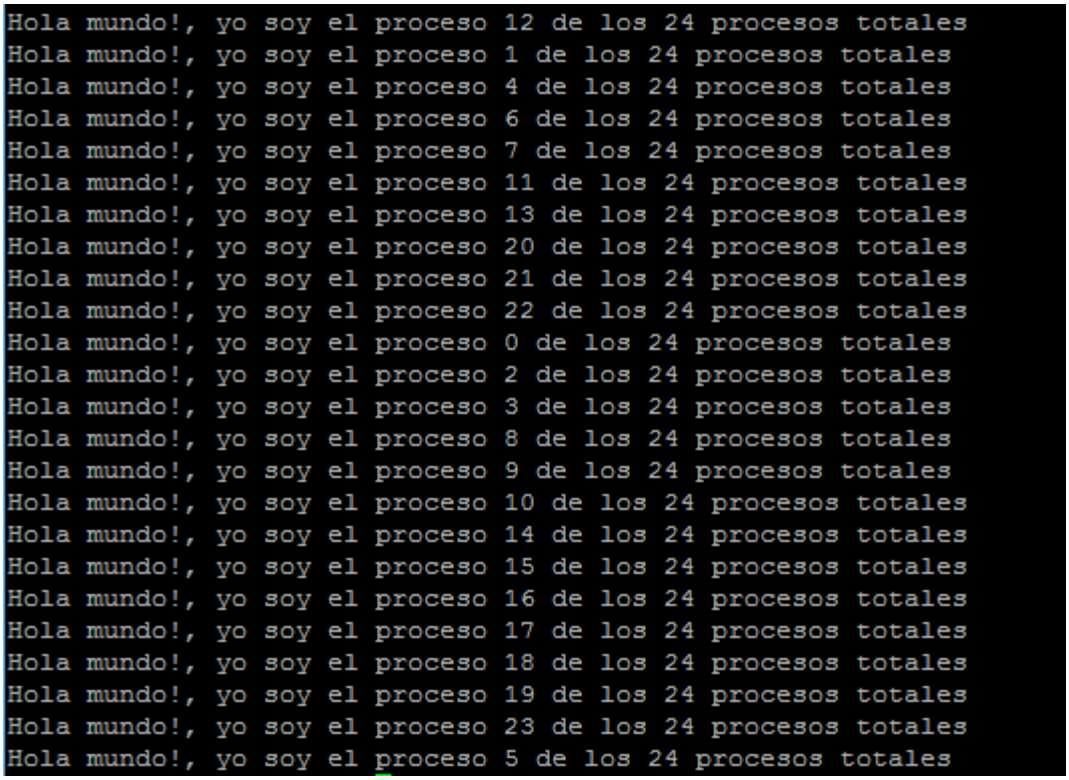


Figura 4.10 (“Ejemplo de resultado de programa paralelo”)

Por lo que cada uno de los procesos, fue ejecutado en cada core del nodo, mencionando que el nodo contiene 24 cores.

Si nosotros eligiéramos 40 procesos, en los parámetros de SLURM, seleccionaríamos 2 nodos, de los cuales utilizará 20 cores de cada uno de los nodos.

Una vez creado este ejemplo, podemos ver claramente, como esto puede ser una excelente herramienta al ejecutar procesos, y muy poderosa, al nosotros poder personalizar los parámetros de ejecución.

4.7. Manejo de cuentas con SLURM

Teniendo claro, todo lo que podemos hacer con el poder de la programación y un software de gestión de procesos en cluster, pasamos a la elaboración de cuentas, para que los usuarios, puedan correr sus respectivos trabajos, además de limitar, Configurar y personalizar las cuentas. Para esto también haremos uso de SLURM.

Para hacer uso de las cuentas con SLURM, debemos haber agregado y configurando correctamente la base de datos, con sus respectivas particiones.

Una vez iniciados los demonios de la base de datos, podemos experimentar con los distintos comandos para la administración de cuentas de SLURM.

Los comandos a seguir son los siguientes:

1.- sacctmgr add cluster fenix
2.- sacctmgr create account name=academic cluster=fénix
3.- sacctmgr create user name=yolixpa account=academic cluster=fénix
4.- scontrol Node=node[5-9] State=Resume
5.-sacctmgr add user Name=yolixpa DefaultAccount=academic Cluster=fenix Partition=amd
6.- sacctmgr modify user yolixpa set GrpCPUs=$cpus where Partition=amd
7.- sacctmgr modify user yolixpa set GrpJobs=${cpus} where Partition=amd
8.- sacctmgr modify user yolixpa set GrpSubmitJobs=${cola} where Partition=amd
10.- scontrol reconfigure

En principio, agregamos el clúster definiendo su nombre, seguido de crear una cuenta, mencionando el cluster, una vez ya creada la cuenta, podemos asignar usuarios, definiendo el nombre, la cuenta y el cluster, y así poder asignar los nodos y el estado.

Una vez creado y asignado el usuario podemos hacer modificaciones acerca de los CPU's y Jobs, por último, con el comando scontrol reconfigure, le decimos al sistema de colas que guarde los cambios nuevos.

CAPITULO 5: “SOFTWARE DE GESTIÓN DE COLAS”

Anteriormente, ya hemos comentado, acerca de distintos softwares que permite la gestión, administración y análisis de las colas en un clúster, también hemos descrito el software actual más utilizado por los clústeres del mundo, “SLURM”, además de este, también existe otros, de distintas cualidades, las cuales nombraremos en este capítulo, tales como PBS, MAUI, TORQUE.

Actualmente muchas supercomputadoras siguen utilizando estos softwares para administrar sus procesos, por lo que en esta tesis mencionaremos las características y cualidad de estos, que son de los más utilizados en el ambiente científico, además la forma de instalación.

5.1. PBS

Actualmente muchas supercomputadoras siguen utilizando estos softwares para administrar sus procesos, por lo que en esta tesis mencionaremos las características y cualidad de estos, que son de los más utilizados en el ambiente científico, además la forma de instalación.

PBS (Portable Batch System) es un sistema flexible de balanceo de carga y planificación de tareas, inicialmente fue desarrollado para administrar recursos computacionales de la NASA. PBS ha sido el líder en la administración de recursos y considerado el estándar de facto para los sistemas de planificaciones bajo sistemas Linux.

En el año de 1986 la NASA, junto al Centro de Investigación Ames desarrolló el primer sistema de manejo de colas para el sistema operativo UNIX, denominado NQS (Network Queueing System). NQS en pocos años se convirtió en un estándar de facto para el manejo de colas por lotes. Una vez que los sistemas paralelos aparecieron, el sistema NQS se volvió inadecuado para manipular requerimientos complejos de administración de recursos.

La NASA lideró un intento por recopilar los requerimientos para un sistema de administración de recursos de siguiente generación. Estos requerimientos y funcionalidades de un sistema para administración de recursos fueron adoptados por la IEEE (Institute of Electrical and Electronics Engineers) en el estándar POSIX 1003.2d. Luego, en el año de 1999 la NASA diseñó un sistema que cumple con los requerimientos del estándar de la IEEE. Este sistema se denominó PBS el cual reemplazó rápidamente a NQS en los supercomputadores tradicionales y sistemas tipo servidor.

La empresa Veridian diseño el sistema PBS para la NASA, y luego, en marzo de 2003, desarrolló una solución integral de administración de balanceo de carga, conocida como PBS Pro, cuya licencia fue adquirida por la empresa Altair Engineering, Inc.

Actualmente existen dos versiones de PBS: una versión antigua de código abierto Altair OpenPBS y la versión comercial Altair PBS Pro. Actualmente OpenPBS y PBS Pro se incluyen en algunos toolkits para la instalación automática de clusters. PBS Pro provee algunas funcionalidades y beneficios para los administradores del cluster.

Las características más importantes del sistema PBS son:

• Múltiples interfaces de usuario: Proveen una interfaz gráfica de usuario para realizar peticiones por lotes y ejecución de tareas.

• Listas de seguridad y control de acceso: El administrador puede permitir o denegar el acceso al sistema PBS basándose en el nombre de usuario, grupo, nodo o dominio de red.

• Registro de tareas: logs detallados de las actividades del sistema mediante el análisis de utilización por usuario, por grupo y por nodo.

• Soporte de tareas paralelas: Permite la utilización de librerías de programación paralela como MPI, PVM, y HPF. Se puede planificar la ejecución de aplicaciones sobre un computador de un sólo procesador o mediante la utilización de múltiples computadores.

• Monitoreo del sistema: Mediante una interfaz gráfica permite realizar un monitoreo completo del ambiente distribuido.

• Soporte para grids: Provee tecnología para grids computacionales y la integración del toolkit Globus.

• API de desarrollo: Permite desarrollar aplicaciones que integran PBS como una de sus herramientas para requerimientos de balanceo de carga.

• Nivel de carga automático: Provee diversas formas de distribuir la carga en los computadores que conforman el cluster, basados en la configuración de hardware, disponibilidad de recursos, actividad del teclado y el manejo de políticas locales.

• Distributed clustering: Permite la disponibilidad de recursos de clusters y otros sistemas distribuidos en una red WAN.

• Ambiente común de usuario: Ofrece al usuario una visión común de las tareas entregadas y solicitadas, el estado del sistema y seguimiento de tareas.

• Prioridad de tareas: Permite a los usuarios especificar prioridades para la asignación de recursos y ejecución de sus tareas.

• Disponibilidad para diferentes plataformas: Permite el soporte para Windows, junto con la mayoría de versiones de UNIX y Linux, desde estaciones de trabajo y servidores hasta supercomputadores.

5.2. OpenLava

OpenLava es un software libre, de código abierto, desarrollado para plataformas IBM® LSFTM, soporta una variedad HPC y aplicaciones analíticas, Con miles de descargas, y cientos de instalaciones, es escalable y robusta, ha sido probada en grupos con miles de núcleos y puestos de trabajo.

Es un software que se controla a través de línea de comandos y de formato de archivo compatible con la mayoría de las características de la plataforma LSF, se puede aprovechar por los cientos de integraciones de software de código abierto y comercial.

Los usuarios o los desarrolladores pueden obtener el código fuente directamente de servidores en la web encargados de manejar versiones u obtener archivos comprimidos que contienen el código fuente de la página principal.

Cuenta con una comunidad activa de usuarios y desarrolladores. [4]

5.3. LoadLeveler

LoadLeveler® para AIX es un sistema de planificación de tareas en paralelo que permite a los usuarios ejecutar más trabajos en menos tiempo, haciendo coincidir las necesidades de procesamiento de cada trabajo y la prioridad con los recursos disponibles, lo que maximiza la utilización de recursos. LoadLeveler también proporciona un punto de control para la gestión eficaz y la carga de trabajo soporta configuraciones de alta disponibilidad. Además, ofrece la contabilidad detallada de la utilización del sistema para el seguimiento o la devolución de cargo.

Cuando los trabajos son sometidos a LoadLeveler, no se ejecutan necesariamente en el orden de presentación. En su lugar, LoadLeveler despacha puestos de trabajo en función de sus prioridades, las necesidades de recursos y las instrucciones especiales; por ejemplo, los administradores pueden especificar que los trabajos de larga duración sólo se ejecutan en fuera de horas, que se programarán los trabajos de corta duración, corriendo por trabajos de larga duración o que los trabajos que pertenecen a ciertos grupos de usuarios tienen prioridad.

Además, los recursos propios pueden ser estrechamente controlados: el uso de máquinas individuales se puede limitar a determinadas horas, usuarios o clases de trabajo o LoadLeveler puede utilizar máquinas sólo cuando el teclado y el ratón están inactivos.

LoadLeveler un seguimiento de los recursos totales utilizados por cada puesto de trabajo en serie o paralelo y ofrece varias opciones de informes de seguimiento de los trabajos y la utilización por el usuario, grupo, cuenta o escriba en un período de tiempo especificado. Para apoyar la devolución de cargo para el uso de los recursos, LoadLeveler puede incorporar velocidad de la máquina para ajustar las tasas de devolución de cargo y ser configurado para requerir una cuenta para cada trabajo.

LoadLeveler soporta configuraciones de alta disponibilidad para garantizar un funcionamiento fiable y monitoriza automáticamente los recursos informáticos disponibles para asegurarse de que no hay tareas programadas para máquinas fallidas.

LoadLeveler capacidades avanzadas y características

Fácilmente escalable en términos del número de nodos de procesamiento en el clúster y el número de puestos de trabajo paralelas en la cola de trabajos, LoadLeveler está en uso en varios de los sitios de supercomputación superiores. LoadLeveler es un producto de calidad industrial que ha estado disponible en plataformas de computación en paralelo durante más de 15 años.

LoadLeveler incorpora las últimas tecnologías en la investigación por lotes programación paralela. Los primeros en adoptar el algoritmo de planificación de relleno, IBM ha continuado mejorando la velocidad, escalabilidad y el rendimiento de este algoritmo en LoadLeveler.

El primer programador de lotes para ofrecer una API de programación completa, LoadLeveler actualmente ofrece una flexibilidad sin precedentes con un conjunto completo de APIs que permiten un alto nivel de personalización específica del sitio.

LoadLeveler es un programa completamente distribuido con amplias capacidades de conmutación por error y auto-reparación de sobrevivir a los eventos del sistema, incluso graves, por lo general sin la intervención del administrador.
LoadLeveler ofrece puntos de control del trabajo y de suspensión con la cancelación de trabajos opcional, mantener y re-cola). Estas capacidades proporcionan una gran flexibilidad en la definición de control de prioridad de trabajo y de recursos en tiempo real.

LoadLeveler se integra con AIX Workload Manager (WLM) para proporcionar tantas especificaciones de recursos en los controles de inicio del trabajo y de utilización de recursos para evitar el uso excesivo de recursos por las aplicaciones errantes.

LoadLeveler se integra fácilmente con InfiniBand adaptadores de canal de host y los conmutadores de grupos de IBM, lo que permite un uso inmediato del ancho de banda escalable y capacidades de multitarea adicionales de los adaptadores de red InfiniBand [5].

5.4. MOAB

Moab, de forma inteligente optimiza el rendimiento de la carga de trabajo y equilibra los niveles de servicio y las prioridades.

La versión básica de de Moab, proporciona la gestión de carga de trabajo además de automatizar la programación, gestión, seguimiento y presentación de informes de las cargas de trabajo de HPC en escala masiva, instalaciones de múltiples tecnologías. El motor de inteligencia Moab patentado utiliza políticas multidimensionales para acelerar

las cargas de trabajo que se ejecutan a través de la combinación ideal de diversos recursos. Estas políticas de equilibrio altas metas de utilización y rendimiento con otras prioridades de la carga de trabajo y los requisitos de SLA. La velocidad y la precisión de las decisiones de planificación automatizados optimiza el rendimiento de carga de trabajo y la utilización de recursos. Esto permite que mayor trabajo realizado en menos tiempo, y en el orden de prioridad derecha. Optimiza el valor y la satisfacción de los sistemas al tiempo que reduce el costo y la complejidad de gestión informática de alto rendimiento (HPC).

Moab, proporciona una gestión de carga de trabajo inteligente que acelera la productividad, automatiza el tiempo de actividad y asegura SLA y se reunió prioridades. Proporciona rápidamente beneficios de gran alcance que impulsan un mayor retorno de la inversión y los resultados de su sistema de computación de alto rendimiento, incluyendo:

- Un mayor rendimiento laboral escalabilidad masiva para una respuesta más rápida y extensibilidad
- La utilización óptima de 90-99% sobre una base consistente, El envío de trabajos sencilla y rápida gestión para aumentar la productividad
- Reducción de los costes de gestión de cluster y de apoyo, así como la complejidad, a través de sistemas heterogéneos errores de trabajos reducidos y de auto-recuperación de los fallos SLA se reunió constantemente para mejorar la satisfacción del usuario y la organización
- La aceleración de la productividad [6].

5.5. OAR

OAR es un recurso y versátil administrador de tareas (también llamado un programador de tareas por lotes) para clusters HPC, y otras infraestructuras de computación (como computación distribuida bancos de pruebas experimentales donde la versatilidad es la clave).

La arquitectura de OAR es basada en una base de datos (PostgreSQL (preferido) o MySQL), un lenguaje de script (Perl) y una herramienta de administración escalable opcional (por ejemplo, Taktuk). Se compone de módulos que interactúan principalmente a través de la base de datos y se ejecutan como programas independientes. Por lo tanto, formalmente, no hay ninguna API, la interacción sistema está completamente definida por el esquema de base de datos. Este enfoque facilita el desarrollo de módulos específicos. De hecho, cada módulo (como programadores) se puede desarrollar en cualquier idioma que tiene una biblioteca de acceso a la base de datos.

Principales características

- Lote y los trabajos interactivos
- Reserva Anticipada
- Normas de admisión

- Adecuación de los recursos (trabajo / propiedades del nodo)
- Pausar y reanudar procesos
- El soporte multi-programadores (FIFO, FIFO simple y con juego)
- Multi-colas con prioridad
- Colas de mejor esfuerzo (para la explotación de recursos ociosos)
- Comprobación de nodos de cálculo antes de arrojar procesos
- Epilogos
- Empleo y recursos las herramientas de visualización.
- Sin Daemon en los nodos de cómputo
- Protocolos de ejecución remota basada en SSH (gestionado por TakTuk)

Optimización del uso de recursos

Ahora, un día, las máquinas con más de 4 núcleos llegan a ser comunes. Por lo tanto, es entonces muy importante ser capaz de manejar núcleos de manera eficiente. Al proporcionar la selección de recursos y procesos de aislamiento en el nivel básico, OAR 2 permite a los usuarios llevar a cabo experimentos que no requieren la exclusividad de un nodo (al menos durante una fase de preparación) para tener acceso a muchos nodos en un solo núcleo, pero dejar el restante núcleos libres para otros usuarios. Esto puede permitir optimizar el número de recursos disponibles.

Al lado, OAR 2 también proporciona una característica de tiempo compartido que permitirá compartir un mismo conjunto de recursos entre los usuarios.
Un acceso más fácil a los recursos

Usando OAR conector 2 OARSH acceder a los recursos del trabajo, usos básicos no requerirán más al usuario configurar su entorno de SSH, ya que todo se maneja internamente (conocido gestión clave de host, etc.). Al lado, los usuarios que realmente no prefieren usar OARSH todavía puede usar SSH con sólo el costo de algunas opciones para configurar (una de las características de la envoltura OARSH es en realidad ocultar estas opciones).
la interconexión recursos grid

Como el acceso a los recursos del clúster uno se limita a un trabajo adjunto, uno puede preguntarse si las conexiones de un trabajo a otro, de clúster a clúster, de sitio en sitio todavía sería posible. OAR 2 proporciona un mecanismo llamado trabajo clave que permite la comunicación entre el trabajo, incluso en varios sitios administrados por varios OAR 2 servidores (este mecanismo se ha utilizado efectivamente por OARGrid2 por ejemplo).

CAPITULO 6: “EXPERIENCIA EN GESTIÓN DE COLAS EN HPC”

A medida que pasa el tiempo, las grandes empresas potencia, fabrican constantemente nuevas tecnologías, a nivel hardware y a nivel software, por lo que, cada vez más rápidamente, tanto la tecnología como el personal, deben actualizarse; El poder de los procesadores, la capacidad de discos de almacenamiento, la cantidad de memoria se escalan a cantidades significativas.

Mencionando así, la primera supercomputadora, introducida en la década de 1970, por la compañía CDC (Control Data Corporation), con capacidades, que en ese entonces era lo más potente, las cuales ahora las podemos encontrar en una computadora convencional, reflexionando, después de 46 años, la tecnología en cuanto a supercómputo, ha crecido a pasos agigantados, teniendo un México, uno de los equipos más potentes a partir del año 2015.

En México, desde el primer supercomputador adquirido por la UNAM, (Universidad Nacional Autónoma de México), hasta estos días, contamos con laboratorios de supercómputo de calidad, como el personal.

6.1. Supercómputo en México.

México, antes de adquirir su primer supercomputador, tenía en distintos institutos, cómputo vectorial, esto significa, varios CPU’s interconectados, con el que, comenzaban bases de programación paralela.

No fue después de 21 años, que se obtuvo la primera supercomputadora en México, adquirida por CRAY, la UNAM creo el primer laboratorio de Supercómputo en México, el sistema operativo era una versión de UNIX, desarrollada por el mismo CRAY, el sistema operativo tenía como nombre UNICOS, mientras las empresas de tecnología continuaban en desarrollo, en México, esta situación perduro por muchos años.

La conexión era posible hacerse remotamente y localmente, los científicos mexicanos, administraban los recursos a través de las utilerías propuestas por el mismo sistema operativo, en este momento, no existía los sistemas de código abierto, por lo que, el personal debía adaptarse a las utilidades que el SO prestaba.

Cabe destacar que cada una de las computadoras desarrolladas, tenían sus propios procesadores, únicos, ósea que, el obtener alguna de estas, era muy caro.

A partir de 1995, muchas de las empresas desarrolladoras, comenzaron a quebrar, y corporaciones como SGI e IBM, comenzaron a tener más poder, al hacer procesadores más baratos y en mayor cantidad, además de sus propios sistemas de administración, los cuales, comenzaban a ser utilizados por más y más computadoras en el mundo, el

primer sistema de administración comercial, fue Load Level, desarrollado por Intel. En este entonces, Intel todavía no se apoderaba del mercado de procesamiento.

Entre 1991 y 1995, en México, hubo 2 supercomputadoras, ambas adquiridas por instituciones de educación, siendo la UNAM y UAM, siendo CRAY.

Mientras tanto, otros estados como San Luis Potosí contaban con cómputo vectorial, apuntando su interés en el Supercómputo.

Entre 1995 y 1997, Intel con PENTIUM, se fue apoderando y se estableció como mercado dominante en cuanto a procesamiento y otros productos de cómputo y en general con HPC, además que, LINUX, también se empezó a apoderar en cuanto a sistema operativo, de más del 50% de las supercomputadoras, para esto, el término “clúster”, comenzó a ser utilizado.

Para 1997, las supercomputadoras de la UNAM y UAM, hacían uno de los mejores cómputos con su procesamiento vectorial con memoria rápida, estimando, que tenía aproximadamente 400MB en disco duro, 4 MB de memoria RAM, teniendo un tamaño físico aproximado de 3 metros por 3 metros.

A partir de aquí, se empezó a imponer los términos de modelo de paralelismo fino, donde el concepto era muchos procesadores, para una sola tarea, los lenguajes dominantes para este tipo de técnicas por excelencia eran c y fortran.

Pronto en México, se empezaron a incorporan productos de marcas japonesas como HITACHI y NECK, que comenzaban a invertir en tecnología de computo vectorial.

Del 2007 al 2010, San Luis Potosí adquirió su equipo de Supercómputo, una supercomputadora CREY xd1, con procesadores opteron AMD, la cual revoluciono por su interconexión por canal optimo desarrollado por Intel, con tecnología HT, (hipertrace) esta supercomputadora se escaló a 900 cores, pero su arquitectura no fue muy eficiente.

En este momento, la administración de los recursos y de los procesos eran optimizando por la aplicación PBS, la cual no era nada estable, y además era muy compleja de utilizar, lo cual orillo a la empresa a comercializar la versión profesional, con licencia de pago, esta versión era estable y también fácil de usar, obligando a los usuarios, a pagar por usar el software.

Del 2010, en adelante, otros Laboratorios empezaron a desarrollarse, con mayor potencia y mayor escabilidad, también de arquitecturas Híbridas, y con distintos tipos de procesadores en el mismo clúster, tal como, el laboratorio nacional de Supercómputo, situado en la Ciudad Universitaria del Estado de Puebla.

Como sugerencia de especialistas en supercómputo en México, sugieren la preparación de personal en todos los niveles, tanto de monitoreo, como administración etc. para que, en un futuro, desarrollemos laboratorios todavía más potentes.

Al año 2016, las instituciones que cuentan con equipos de Supercómputo son:

- BUAP
- IPICyT
- UNAM
- CICESE
- UNIVERSIDAD DE SONORA
- UAEMEX
- UDG
- UNACH
- UAM
- CINVESTAV

Como podemos ver, a lo largo de la historia de Supercómputo en México, los sistemas de colas, han pasado por distintas fases, en principio, los sistemas de colas fueron generados por las corporaciones que fabricaban los equipos de supercómputo, en caso de México, el primer fabricante CREY, en su apartado de utilerías, los científicos podían organizar y ejecutar sus procesos a través de su propio sistema de colas.

Después, cuando los sistemas de colas empezaron a comercializarse, ya sea en su forma libre o en su forma de licencia, LoadLeveler, fue uno de los pioneros en sistemas de colas en México.

En la actualidad, los sistemas de colas que son utilizados en México, son: SLURM, PBC versión libre y versión con licencia, LoadLeveler y Moab.

CAPITULO 7: “EVALUACIÓN E INSTALACIÓN DE HERRAMIENTAS DE GESTION DE COLAS”

A continuación, procederemos a describir detalladamente la instalación de los sistemas de colas más utilizados a nivel global.

7.1. Instalación PBS

Antes de hacer la instalación de PBS, debemos dar permisos a los nodos conectarse remotamente a través de protocolos de comunicación, tales como, ssh, rsh, etc. A preferencia del usuario, se con de la manera más óptima para la utilización de este software.

En principio, debemos de descargar el Software OpenPBS, la versión de código abierto de PBS, el cual podemos obtener de la liga www.openpbs.org, registrarte ante este sitio es completamente requerido.

1.- Descomprimir PBS

```
# cd /home/download
# tar -xvzf OpenPBS_2_3_16.tar.gz
# cd OpenPBS_2_3_16/
```

7.1 ("Descompresión PBS.”)

2.- Parchar los archivos PBS, hay que descargar los parches desde la misma página mencionada.

```
# cd /home/download/OpenPBS_2_3_16
# patch -p1 -b < pbs.patch
```

7.2 ("Parchar archivos PBS.”)

3.- Compilar PBS.

A. Head Node

```
# mkdir /home/downlad/OpenPBS_2_3_16/head
# cd /home/downlad/OpenPBS_2_3_16/head
# ../configure --disable-gui --set-server-home=/var/spool/PBS --set-default-
server=node00
# make
# make install
```

This disables the GUI and sets the installation directory to /var/spool/PBS instead of /usr/spool/PBS. This also sets the default PBS server to node00, which should be the hostname of the head node. The source is then compiled and the binaries are then installed.

B. Compute Nodes

```
# mkdir /home/downlad/OpenPBS_2_3_16/compute
# cd /home/downlad/OpenPBS_2_3_16/compute
# ../configure --disable-gui --set-server-home=/var/spool/PBS --disable-server
--set-default-server=node00 --set-sched=no
# make
# rsh nodeXX 'cd /home/download/OpenPBS_2_3_16/compute; make install'
```

After the head node is successfully installed, the configuration script is run again in a separate directory for the compute nodes. This disables the GUI and sets the installation directory to /var/spool/PBS instead of /usr/spool/PBS. This also sets the default PBS server to node00, which should be the hostname of the head node. It also disables the server and scheduling processes of PBS on the compute nodes, since they are not necessary. The source is then compiled and installed using rsh.

Figura 7.3 ("Compilar PBS")

4.- Instalar la documentación.

```
# cd /home/download/OpenPBS_2_3_16/head/doc
# make install
```

Figura 7.4 ("Instalar documentación")

Configuración PBS

5.- Crear el archivo de descripción de nodos en PBS.

```
#This file contains a list of all of the compute nodes hostnames
node01 np=1
node02 np=1
      -
      -
nodeXX np=1
```

Figura 7.5 ("Configurar PBS")

Donde, nodexx, corresponde al hostname del nodo de cómputo y np corresponde al número de procesadores con los que cuenta el nodo.

6.- Configurar PBS mom, creando el archivo en /var/spool/PBS/mom_priv/config.

```
#/var/spool/PBS/mom_priv/config
$logevent 0x0ff
$clienthost node00
$restricted node00
```

Figura 7.6 ("Configurar PBS mom")

Esto hace que todos los mensajes excepto para la depuración, sean registrados y establece el servidor primario al nodo 00. También permite que el nodo 00 supervise OpenPBS. En el nodo principal y en cada nodo de cálculo, comenzar con PBS mom.

7.- Configurar servidor PBS.

En el nodo cabecera.
/usr/local/sbin/pbs_server –t create
qmgr

```
>c q workq
>s q workq queue_type=execution
>s q workq enabled=true
>s q workq started=true
>s s default_queue=workq
>s s scheduling=true
>s s query_other_jobs=true
>s s node_pack=false
>s s log_events=511
>s s scheduler_iteration=600
>s s resources_default.neednodes=1
>s s resources_default.nodect=1
>s s resources_default.nodes=1
>quit
```

Figura 7.6 ("Configurar servidor PBS")

Esto crea una cola de ejecución, llama al trabajo habilitado y lo inicia. A continuación, se declara la cola predeterminada para el servidor. La sesión y programación habilitados en el servidor y el paquete se establece en false. El número predeterminado de nodos se establece en 1.

Es útil para la copia de seguridad del archivo de configuración del servidor con PBS.
#qmgr –c “Servidor de Impresión” > /var/spool/PBS/qmgr.conf, para que el servidor PBS pueda ser restaurado con:

#qmgr < /var/spool/PBS/qmrg.con
8.- Iniciar el programa PBS.

En el nodo cabecera, la programación PBS, necesita ser iniciada después de que el servidor de configuración está completo. Esto lo hacemos de la siguiente manera:

#/usr/local/sbin/pbs_sched

9.- Activar PBS.

En el nodo cabecera:

A.- Crear el script /etc/init.d/pbs
B.- Configurar PBS, para que se restaure cuando la computadora se reinicie. #chckconfig pbs on

10.- Restaurar PBS.

En el nodo cabecera y nodos de computo, restaurar manual mente OpenPBS, así, todos los archivos de configuración en los nodos, surtirán efecto.

#/sbin/service pbs stop
#/sbin/service pbs start

Probando PBS

Después de haber configurado PBS, podemos correr un script simple y verificar que todo éste trabajando correctamente.

```
#!/bin/sh
#testpbs
echo This is a test
echo Today is `date`
echo This is `hostname`
echo The current working directory is `pwd`
ls -alF /home
uptime
```

Figura 7.7 ("Test PBS")

Por último, con el comando “qsub testpbs” mostraremos el estado del proceso y la cola.

7.2. Instalación MAUI / TORQUE

Comenzaremos descargando el archivo de instalación desde el link http://www.adaptivecomputing.com/support/download-center/torque-download/. Aquí, podemos encontrar todas las versiones.

1.- Descomprimir TORQUE y dirigirnos a la carpeta correspondiente.

```
> tar -xzvf torque-4.1.3.tar.gz
> cd torque-4.1.3/
```

Figura 7.8 ("Instalación Torque")

2.- Configurarla instalación, para personalizar esta configuración hay diferentes parámetros, que podemos encontrar en: http://docs.adaptivecomputing.com/torque/4-1-3/Content/topics/1-installConfig/customizingTheInstall.htm

```
> ./configure
```

Figura 7.9 ("Configuración Instalación")

Por default, el comando make install, instala todos los archivos en, /usr/local/bin, /usr/local/lib, /usr/local/include, /usr/local/man, puedes especificar un prefijo de instalación que no sea /usr/local/ mediante el uso de –prefix, como un argumento a ./configure.

3.- Ejecutar comandos, make y make install.

```
> make
> sudo make install
```

Figura 7.10 ("Compilar")

4.- Después de la instalación, comprueba que tiene la variable de entorno PATH, configurando para incluir /usr/local/bin y /usr/local/sbin/local, por defecto, cuando se realiza la instalación, crea un directorio en, /var/spool/par. Este directorio se conoce como TORQUE_HOME. Este, tiene varios subdirectorios, incluyendo server_priv, server_logs, mom_priv, mom_logs, y otros directorios utilizados en la configuración, para el funcionamiento de PAR.

5.- Comprobar que se han configurado las variables de entorno para que el sistema pueda encontrar las librerías compartidas y archivos binarios para el PAR.

Para establecer la ruta de la librería, añadir el directorio donde se instalan las librerías de par. Por ejemplo, si se instalan las bibliotecas de PAR, en /local/bin/lib/usr ejecute lo siguiente:

```
> echo '/usr/local/lib' > /etc/ld.so.conf.d/torque.conf
> ldconfig
```

Figura 7.11 ("Ejemplo de establecer la ruta de líbreria")

6.- Habilitar como servicio

- **Red Hat (as root)**

```
> cp contrib/init.d/pbs_mom /etc/init.d/pbs_mom
> chkconfig --add pbs_mom
```

- **SuSE (as root)**

```
> cp contrib/init.d/suse.pbs_mom /etc/init.d/pbs_mom
> insserv -d pbs_mom
```

- **Debian (as root)**

```
> cp contrib/init.d/debian.pbs_mom /etc/init.d/pbs_mom
> update-rc.d pbs_mom defaults
```

Figura 7.11 ("Habilitar como servicio")

7.3. Instalación SLURM

Antes de proceder con la instalación de SLURM, algunas dependencias ya deben estar instaladas en nuestro servidor, en este caso, utilizaremos cent OS 6.8

Las dependencias básicas son las siguientes:

yum install hwloc hwloc-devel numactl readline-devel mysql-devel pam-devel perl-ExtUtils-MakeMaker rrdtool freeipmi lua-devel gtk2-devel redhat-lsb redhat-rpm-config -y

De ahí, las siguientes aplicaciones.

Descarga de dependencias.

Las dependencias que instalaremos son BLCR (Berkeley Lab Chekpoint / Restart), una tecnología de núcleo híbrido, de punto de control / reinicio, con el cual no es necesario agregar o editar el archivo fuente del núcleo del sistema operativo, esto asegura de

que además de que SLURM funcione correctamente, otra gama de aplicaciones con distintos requisitos.

La segunda dependencia que ocuparemos es, Munge, una aplicación para la creación de llaves encriptadas, lo cual asegura que SLURM, tenga un nivel alto de seguridad.

Comenzaremos con la instalación de BLCR.

Instalación de BLCR

Esta dependencia requiere aplicaciones básicas para su instalación, a continuación, con el siguiente comando, las instalaremos:

```
# yum install wget gcc gcc-c++ make kernel-devel kernel-headers perl rpm-build –y
```

Una vez seguros de que estas fueron instaladas correctamente, nos queda proceder con la descarga del archivo fuente de BLCR.

```
# wget http://crd.lbl.gov/assets/Uploads/FTG/Projects/CheckpointRestart/downloads/blcr-0.8.5.tar.gz
```

Una vez obtenido los archivos fuentes, compilamos con código con el siguiente comando:

```
# rpmbuild -tb --define 'with_multilib 0' blcr-0.8.5.tar.gz
```

Y para finalizar, instalamos los paquetes, dirigiéndonos al directorio de instalación:

```
# cd /root/rpmbuild/RPMS/x86_64
# yum install blcr* --nogpgcheck –y
```

Con esto, terminamos la instalación de BLCR.

Instalación de Munge

Para comenzar la instalación debemos descargar los archivos fuente de munge:

```
# wget https://munge.googlecode.com/files/munge-0.5.11.tar.bz2 --no-check-certificate
```

Compilamos los archivos con el siguiente comando:

```
# rpmbuild -tb --clean munge-0.5.11.tar.bz2
```

Instalamos los paquetes:

```
# cd /root/rpmbuild/RPMS/x86_64/
# yum install munge* -y
```

Una vez instaladas estas dependencias, procedemos a la instalación de SLURM.

1.- En principio, debemos obtener los archivos fuente de SLURM.

```
# wget http://www.schedmd.com/download/latest/slurm-14.03.8.tar.bz2
```

2.- Compilamos los RPMs

```
# rpmbuild -tb --with mysql --with blcr slurm-14.03.6.tar.bz2
```

3.- Instalamos los paquetes de SLURM.

```
# yum install slurm* -y
```

4.- Creamos la llave secreta de MUNGE.

```
# dd if = /dev/random bs=1 count=1024 > /etc/munge/munge.key
```

Nota: Estos comandos pueden estar sujetos a cambios dependiendo de la versión de Linux, la lógica es la misma al igual que las dependencias, en algunos cambian las formas de ser compilado.

“CONCLUSIONES Y ANOTACIONES”

A futuro, se espera el crecimiento de potencia en nodos, por lo que la administración y gestión de colas en cómputo de alto rendimiento, será muy importante, sugiriendo a los desarrolladores de software para este crecimiento, no solo hacer pruebas a un alto nivel escalable, si no, adecuar el nuevo software a el hardware de nueva generación y explorar nuevos sistemas de comunicación, agilizando el procesamiento de procesos paralelos junto con el trabajo del software de gestión de colas.

El fin de esta tesis es conocer, el tema de especialidad acerca de la administración de procesos en equipos de supercómputo, ya que es un tema relativamente nuevo.

Como inicio pensando a futuro, sugiero el estudio de la inyección de un proceso a otro proceso, de esta manera, se podría experimentar a nivel de inteligencia artificial, ya que siendo así, el recorrido de árboles de decisión, sería más rápido, obteniendo una respuesta más ágil. Y en muchos campos más, esto sería una buena iniciativa de evolución en cuanto a procesamiento.

En esta tesis había intención de explorar software de licencia, lo cual no sucedió por motivos los cuales, no fue posible adquirir la licencia privada.

Este es solo un punto importante para la administración de un cluster como tal, además, la operación de cluster, requiere un manejo adecuado de los recursos asociados. Los recursos deben ser administrados adecuadamente, invertir la menor cantidad de tiempo, en detectar, investigar y recuperar fallos, tanto de hardware como de software, y de este modo, definir las posibles medidas de contingencia y tratar que el sistema esté libre de errores. A su vez, estos pasos permiten la adaptabilidad a los requerimientos y cambios constantes que se presentan en la manipulación de tecnologías cluster, en cuanto se refiere al hardware, software y al uso de ciertos patrones de diseño.

BIBLIOGRAFÍA

[1] Slurm Workload Manager. *Documentación SLURM.* [Consulta: 27 de abril de 2016]. Disponible en web: http://slurm.schedmd.com

[2] Adaptive Computing. *Documentación PBS.* [Consulta: 27 de abril de 2016]. Disponible en web: http://docs.adaptivecomputing.com/torque/4-1-3/Content/topics/1-installConfig/architecture.htm

[3] Escuela Politécnica Nacional de Ecuador*. Definición de administración de cluster*, [Consulta: 27 de abril de 2016]. Disponible en web: http://clusterfie.epn.edu.ec/clusters/Definiciones/definiciones4.html

[4] Open Source Workload Management. *About OpenLava,* [Consulta: 28 de abril de 2016]. Disponible en web: http://www.openlava.org/home.html

[5] IBM. About LoadLeveler [Consulta: 28 de abril de 2016]. Disponible en web: http://www-03.ibm.com/systems/power/software/loadleveler/

[6] Adaptive Computing. About MOAB HPC Suite Basic Edition [Consulta: 28 de abril de 2016]. Disponible en web: http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/

[7] Oar. About OAR [Consulta: 28 de abril de 2016]. Disponible en web: https://oar.imag.fr/start