PROGRAMACIÓN DE ARQUITECTURAS PARALELAS

José M. Cámara
(checam@ubu.es)
v. 2.0

UNIVERSIDAD DE BURGOS

UNIVERSIDAD DE BURGOS

Guión

Niveles de aproximación.

Nivel de sistema operativo.

- Sistemas Operativos de red.
- Sistemas Operativos multiprocesador.
- Sistemas Operativos distribuidos.
  - Sistemas de paso de mensajes.

Nivel de aplicación.

- Programación concurrente.
  - Sincronización en variable compartida.
  - Sincronización en paso de mensajes.
  - Gestión de procesos concurrentes.
  - Motivación para la programación concurrente.

Nivel de gestión.

- Planificación.
- PBS.

Conceptos relevantes

Programa: código a ejecutar por el procesador (nivel de máquina).

Usuario: persona que hace uso de la máquina (programador,administrador,…)

Nivel de sistema operativo

Sistemas operativos para multiprocesadores con memoria compartida (SMP):

- Software **fuertemente** acoplado
- sobre Hardware **fuertemente** acoplado

Sistema operativo de red (actualmente todos):

- Software **débilmente** acoplado
- sobre Hardware **débilmente** acoplado

Sistema operativo distribuido (SOD):

- Software **fuertemente** acoplado
- sobre Hardware **débilmente** acoplado

La tradicional multiplicidad de sistemas disponibles (normalmente versiones propietarias de UNIX), ha venido a converger en un limitado número de sistemas UNIX (LINUX casi siempre) y la irrupción de Windows.

Sistemas operativos de red.

Permiten realizar operaciones en máquinas remotas a través del adaptador de red.

En la actualidad, todos los sistemas operativos ofrecen esta funcionalidad, por lo que no vamos a profundizar en su estudio.

Se trata de una funcionalidad que permite la computación paralela, pero de una manera muy poco eficiente.

Sistemas operativos multiprocesador I

Al igual que los sistemas operativos de red, proporcionan una funcionalidad que actualmente está disponible por defecto.

Permiten la gestión de un entorno de multiprogramación sobre un hardware multiprocesador.

Respecto a la gestión monoprocesador, la diferencia está en que hay un recurso, el procesador, que se ha multiplicado.

El planificador debe dar servicio a la cola de procesos, haciendo uso eficiente de los diferentes procesadores disponibles.

Su diseño les permite gobernar tanto entornos multiprocesador como multinúcleo o ambos.

Sistemas operativos multiprocesador II

Dos conceptos relevantes:	Proceso: unidad de protección y asignación de recursos.
	Hilo: unidad de planificación.

Un proceso puede albergar varios hilos, cada uno de ellos con su propia pila, estado de ejecución y contexto del procesador.

Los hilos a planificar pertenecen a las aplicaciones de usuario y al propio sistema operativo. A este respecto tenemos:	Multiprocesadores simétricos: todos los nodos tienen la misma funcionalidad y pueden ejecutar igualmente el SO.
	Multiprocesadores asimétricos (maestro – esclavo): un nodo ejecuta SO y otros solamente aplicaciones -> Si el nº de esclavos es alto, el maestro se colapsa.

Un problema a atajar es la posible aparición de condiciones de carrera en el acceso a la cola de procesos por parte de un planificador ejecutado concurrentemente en varios procesadores. Para evitarlo, el SO se ejecutaría en exclusión mutua.

Sistemas operativos multiprocesador III - Planificación

En los multiprocesadores la planificación de procesos (hilos) se convierte en una tarea doble:

- Decidir qué proceso recibe tiempo de CPU.
- Decidir qué CPU alberga el proceso.

Existen diferentes modelos:

- Tiempo compartido: todos los hilos se encolan y son planificados de forma independiente.
- Espacio compartido: los hilos relacionados (pertenecientes a un mismo proceso o tarea) se planifican en conjunto, empiezan cuando hay disponible un número suficiente de CPUs y no son interrumpidos).
- En pandilla (gang): los hilos pueden ser interrumpidos, pero los que están relacionados son interrumpidos y recuperados a la vez.

Sistemas operativos multiprocesador IV - Ejemplo

LINUX incorpora soporte para multiprocesadores desde la versión 2.6 de su núcleo. En ella implementa el planificador O(1).

La planificación de procesos se realiza en tiempo compartido, para lo cual mantiene una cola de procesos activos y una cola de procesos expirados por CPU.

Los procesos que han consumido su “slot” de tiempo asignado pasan a la cola de expirados.

Para un determinado nivel de prioridad, cuando todas sus tareas se encuentran expiradas, ésta cola pasa a ser activa.

Cada cola soporta 140 niveles de prioridad, siendo los 100 más altos para tareas de tiempo real. En cada nivel, la cola de procesos es FIFO.

Cada 200ms se monitoriza la carga de los procesadores y se rebalancea.

La existencia de una cola por CPU permite mejorar la eficiencia de cache. La recuperación de una tarea previamente procesada, disminuye los fallos de cache.

Sistemas operativos distribuidos

Logran que un hardware formado por computadores independiente, funcione a ojos del usuario como una sola máquina (máquina virtual).

- Los SO actuales no proporcionan esta funcionalidad.

Existen diversas formas de implementar el sistema:

- Arquitectura cliente servidor: implementadas por ejemplo en sistemas de información en los que clientes locales acceden a servidores remotos mediante un lenguaje de consultas.
- Llamadas a procedimiento remoto: las aplicaciones invocan procedimientos que se ejecutan en máquinas remotas.
- Sistemas de paso de mensajes: procesos ejecutados en máquinas diferentes se comunican entre sí mediante mensajes y colaboran para realizar una tarea. Se trata de la opción por defecto en sistemas de computación paralela.

Para lograrlo va a ser necesaria una capa adicional de software denominada middleware que proporciona la funcionalidad de máquina virtual, que está basada en dos conceptos:

- Migración de procesos: debe ser capaz de mover procesos de una máquina a otra en tiempo de ejecución para, entre otras cosas, lograr un balanceo dinámico de carga.
- Tolerancia a fallos: debe responder sin fallos a situaciones de incorporación o desaparición de componentes de la máquina.

UNIVERSIDAD DE BURGOS

Sistemas de paso de mensajes

Alternativas del sistema de paso de mensajes:

- Fiable: garantiza la entrega del mensaje y dispone de mecanismos de notificación al emisor.
- No fiable: entrega en mensaje a la red y lo olvida. Es menos seguro pero menos costoso en términos de sobrecarga de la red.
- Bloqueante: interrumpe al proceso peticionario hasta que se haya completado la operación requerida (envío o recepción). Considera que se ha completado la operación cuando la información se encuentra en un buffer local, no cuando ha llegado a destino en caso de envío.
- No bloqueante: continúa el procesamiento independientemente del estado del mensaje. El envío o recepción se procesa en background. Permite acelerar el procesamiento pero puede producir condiciones de carrera.
- Síncrono: interrumpe al emisor hasta que el receptor ha recogido el mensaje.
- Asíncrono: solamente interrumpe al emisor en función de si el sistema es bloqueante o no.

La implementación del middleware puede tomar una decisión en cuanto a fiabilidad, bloqueo y sincronización, pero también puede proporcionar varias alternativas.

El middleware se puede implementar bajo dos estrategias diferenciadas:

- Extensión del sistema operativo que le proporciona la funcionalidad de sistema distribuido
- Capa intermedia independiente del SO que proporciona un API a las aplicaciones de usuario (MPI).

Ambas opciones pueden ser complementarias: la primera de ellas suele responder mejor al modelo de máquina virtual, mientras que la segunda facilita el desarrollo de aplicaciones paralelas mientras que implementa los servicios mínimos para proporcionar soporte de máquina virtual.

Nivel de aplicación

Dos paradigmas en cuanto al desarrollo de aplicaciones paralelas:	Paralelismo implícito: el usuario no se implica en el aprovechamiento de las posibilidades del hardware. Confía en que las capas inferiores (compilador, sistema operativo) lo hagan por él.
	Paralelismo explícito: el usuario desarrolla sus aplicaciones bajo alguna de las alternativas que permiten la explotación de un hardware paralelo.
Modelos de paralelismo explícito:	Lenguajes tradicionales con librerías paralelas (MPI). Solución simple en cuanto a esfuerzo inicial, pero con alta implicación por parte del usuario. Un programador inexperto puede provocar una pérdida de rendimiento.
	Extensiones a lenguajes tradicionales. Requieren de un nuevo compilador y un esfuerzo de aprendizaje algo mayor.
	Directivas de compilación: el paralelismo se explicita a nivel de directivas que, si se obvian, no impiden que el programa se compile como secuencia.

Una vez más, existen soluciones que combinan varias alternativas. Por otro lado, el esfuerzo de aprendizaje puede depender de su ámbito de aplicación (CUDA).

Programación concurrente

Concepto: se produce cuando el programador se encarga de forma explícita de la creación, destrucción y sincronización hilos, procesos o tareas.

Cada una de esas entidades se va a ejecutar en un procesador virtual.

Si el procesador virtual es también real, hablaremos de programación paralela.

Independientemente del entorno en que se desarrolle, el entorno de programación debe proporcionar una serie de servicios:	Expresión de la ejecución concurrente mediante el concepto de proceso (hilo o tarea según los casos).
	Comunicación entre procesos.
	Sincronización entre procesos.
Los procesos pueden ser:	Independientes: no necesitan comunicación ni sincronización.
	Cooperativos: se comunican y sincronizan para realizar una labor común.
	Competitivos: son independientes pero compiten por recursos comunes para lo cual van a requerir comunicación y/o sincronización.
La comunicación puede ser:	Variable compartida.
	Paso de mensajes.

Sincronización en variable compartida I

Motivo: evitar condiciones de carrera en el acceso a secciones críticas.

Sección crítica: zona de código en la que se modifica una variable compartida.

Opciones:

- Espera ocupada:
- Semáforos:
- Regiones críticas condicionales:
- Monitores:
- Objetos protegidos:
- Métodos sincronizados:

Sincronización en variable compartida II

Espera ocupada: antes de entrar en una sección crítica el proceso comprueba si hay "alguien" dentro de ella. Para ello será necesario implementar un indicador y sincronizar su gestión. La complejidad de la sección crítica debe justificar este sobreesfuerzo. Se puede llevar a cabo mediante un indicador por proceso y una variable turno. El indicador señala qué procesos intentan acceder. Si hay más de uno, decide la variable turno. Cuando el proceso acaba, cambia el turno al siguiente.

- Es un procedimiento que escala mal a un número elevado de procesos.
- Mientras un proceso espera su turno, se encuentra comprobando el turno constantemente, lo que supone una penalización de rendimiento.
- Esto se podría mejorar con un mecanismo de suspensión y reanudación.

Semáforos: son variables enteras no negativas manejadas mediante procedimientos: wait(decrementa el valor si es >0 y si no, espera) y signal(incrementa el valor). Ambas operaciones son atómicas. Los procesos en espera son encolados y suspendidos. La gestión de la cola admite diversas alternativas (FIFO por defecto).

Sincronización en variable compartida III

Regiones críticas condicionales: son zonas de código que sólo se pueden ejecutar en exclusión mutua. Los procesos en espera despiertan periódicamente para comprobar la condición de acceso a la que se accede en exclusión mutua también.

Monitores: módulos que encapsulan las regiones críticas de un programa. No facilita mecanismos de sincronismo adicionales, pero encapsula todas las operaciones.

Objetos protegidos, métodos sincronizados: herramientas adicionales proporcionadas por entornos determinados (Ada, Java) para facilitar la sincronización.

Sincronización en paso de mensajes

Relacionado con lo explicado para middleware:

- Sistemas síncronos.
- Sistemas asíncronos.
- Invocación remota: el emisor continúa cuando ha recibido una respuesta por parte del receptor.

Gestión de procesos concurrentes I

Estructura: afecta a la creación de los procesos:	**Nivel:**	**Granularidad:**	**Inicialización:**
• Número fijo de procesos. • Número variable en tiempo de ejecución.	• Modelo plano. • Procesos anidados.	• Grano grueso: pocos procesos realizan labores complejas. • Grano fino: muchos procesos realizan labores simples.	• La información se pasa a los procesos en su arranque. • La información se pasa a los procesos en tiempo de ejecución.

Gestión de procesos concurrentes II

Explicitación de la creación de procesos:

- Fork/join: permite crear procesos dinámicamente e inicializarlos mediante paso de parámetros. Los procesos hijos son finalizados por el padre. Se trata de un mecanismo propenso a errores. (C/Pthreads).
- Cobegin: ejecución concurrente de una secuencia (hasta que aparece un coend). Modelo plano. Todos los procesos menos uno mueren cuando se acaban sus instrucciones.
- Declaración explícita: el propio lenguaje proporciona estructuras para dar de alta procesos.

Finalización (cómo mueren los procesos):

- Finalización del cuerpo de ejecución: se terminan las instrucciones.
- Suicidio: autofinalización.
- Aborto por parte de otro proceso.
- Situación de error sin tratar.
- Nunca (aplicaciones embebidas).
- Cuando no son necesarios (un proceso servidor al que se le han muerto los clientes).

Motivación para la programación concurrente

Modelo ajustado a la realidad: el mundo es concurrente.

Permite incrementar el rendimiento de sistemas paralelos.

Permite incrementar el rendimiento de sistemas no paralelos (ejemplo del piloto automático).

UNIVERSIDAD
DE BURGOS

PLANIFICACIÓN (SCHEDULING)

Gestión de trabajos de usuario en supercomputadores

Introducción

La planificación (scheduling) de trabajos en máquinas paralelas consiste en decidir qué trabajos de los que se encuentran en espera de ser atendidos, son conducidos a ejecución en cada momento y en qué condiciones.

El impacto de este aspecto de la computación en el rendimiento global del sistema es crítico.

No existe un procedimiento algorítmico para determinar la mejor solución posible, por lo que las alternativas se disparan.

Se trata de un problema enormemente complejo dados los múltiples factores que intervienen.

El enunciado del problema ha ido cambiando en los últimos años debido a la desaparición de ciertos tipos de máquinas paralelas y la aparición de otras nuevas.

En los últimos tiempos, la práctica totalidad de las máquinas son de memoria distribuida, ganando terreno cada vez más los clústeres. En este escenario, la memoria ya no se trata como un recurso independiente, sino que queda ligada a la asignación de los procesadores a los que se encuentra asociada.

Conceptos asociados

Partición:

- se trata del conjunto de recursos asociados a un trabajo. El propio concepto se asocia a la existencia de máquinas con un grado de acoplamiento medio – alto.
- En la actualidad se va sustituyendo por otro tipo de conceptos como “chunk” en PBS.

Precedencia (preemtion):

- capacidad de suspender procesos en curso para atender a otros que se consideran más prioritarios.

Alternativas I

Especificación de particiones (asignación de recursos):

Estática: la asignación de recursos se mantiene durante la ejecución de las aplicaciones.	Fija: el número de procesadores es determinado por el administrador y no puede ser modificado.
	Variable: la decisión se basa en la petición de recursos que hace el usuario al enviar el trabajo.
	Adaptativa: la asignación la determina el planificador cuando inicia el trabajo. Se basa en la carga del sistema y la petición del usuario.

Dinámica: la cantidad de recursos se puede modificar en tiempo de ejecución en función de las condiciones de carga del sistema, prioridad, etc.

Alternativas II

Flexibilidad de los trabajos:

Trabajos rígidos:	el número de procesadores asignados a un trabajo se determina externamente al planificador y permanece inalterable en tiempo de ejecución.
Trabajos moldeables:	el número de procesadores lo determina el planificador con ciertas restricciones y este número permanece constante durante la ejecución.
Trabajos evolutivos:	el trabajo atraviesa fases en las que requiere un número de recursos diferente, de manera que el número de procesadores asignados puede cambiar en base a esos requerimientos.
Trabajos maleables:	el número de procesadores asignados puede cambiar en tiempo de ejecución a voluntad del planificador, ya sea incrementando, o decrementando los recursos asignados en función de las necesidades del conjunto.

Alternativas III

Nivel de precedencia permitido:

Sin precedencia:	los trabajos no pueden ser interrumpidos una vez iniciados.
Precedencia local:	los hilos de un trabajo pueden ser interrumpidos, pero posteriormente continuarán en el mismo procesador.
Precedencia migrable:	un hilo suspendido en un procesador puede ser reanudado en otro.
Planificación en grupo:	todos los hilos de un mismo trabajo son suspendidos y reanudados a la vez, con o sin migración.

Políticas

Existen diferentes objetivos posibles en el ánimo del administrador:

- Minimizar el tiempo de espera de los procesos en cola.
- Minimizar el tiempo de ejecución.
- Maximizar el throughput del sistema.
- Maximizar el grado de utilización del sistema.

Consideraciones finales

La asignación dinámica de recursos tiende a proporcionar mejores resultados independientemente de los objetivos perseguidos.

En contrapartida, supone una mayor complejidad, ya que el planificador debe obtener información del estado de los trabajos y los recursos en tiempo de ejecución así como implementar los algoritmos de toma de decisiones.

Aunque la disminución del tiempo de espera en cola no sea el objetivo principal, un mecanismo ampliamente implementado es el denominado “backfilling” que consiste en aprovechar recursos reservados para un trabajo que no ha podido reunir aún todos los necesarios, para ejecutar trabajos más pequeños que se prevé que finalicen antes de que el proceso en espera acabe de obtener todos los recursos que necesita para arrancar.

UNIVERSIDAD
DE BURGOS

PBS

Gestión de colas y planificación de trabajos

Introducción

PBS (Portable Batch System): sistema de gestión de cargas de trabajo desarrollado originalmente para gestionar los recursos de computación de la NASA.

Proporciona una interfaz de acceso uniforme para que los usuarios puedan “encolar” sus trabajos.

Proporciona herramientas administrativas para que los administradores puedan tener control de sus recursos de computación.

Se trata de un producto propietario pero que se ajusta al estándar IEEE 1003.2d

Implementa los 3 elementos básicos de cualquier sistema de gestión de cargas de trabajo:	Gestión de colas: permite recoger los trabajos generados por los usuarios y “encolarlos” en espera de disponer de los recursos necesarios para su procesamiento.
	Planificación (scheduling): permite establecer políticas de selección de trabajos y asignación de recursos.
	Monitorización: permite conocer el grado de utilización de los recursos para poder tomar las decisiones adecuadas.

Arquitectura de PBS

Comandos

- permiten a los usuarios acceder a la funcionalidad del sistema:
  - De usuario: permiten a los usuarios del sistema enviar trabajos y monitorizar su evolución.
  - De administrador: permiten la gestión completa del sistema.
  - De operador: permiten una gestión limitada del sistema.

Servidor (de trabajos)

- proporciona los servicios básicos para el procesamiento de trabajos.

Ejecutor (de trabajos)

- se encarga de poner los trabajos en ejecución.
- devuelve la salida del trabajo al usuario cuando sea requerido.
- corre en cada una de las máquinas que ejecutan trabajos.

Planificador

- implementa la política de selección de trabajos y asignación de recursos.
- se comunica con el ejecutor para conocer la disponibilidad de recursos y con el servidor para conocer la demanda de trabajos pendientes.

Terminología

Nodo (obsoleto)	• sistema de computación con sistema operativo único y espacio de memoria virtual unificado.
Vnodo (nodo virtual)	• unidad de computación utilizable por el sistema. • es la unidad de asignación de recursos.
Host	• maquina con su propio sistema operativo dotada de uno o varios recursos de computación (Vnodos).
Chunk	• conjunto de recursos asignados a un trabajo.

Cola	• contenedor de trabajos dentro de un servidor:
	  • De rutado: utilizada para mover trabajos a otras colas.
	  • De ejecución: en la que debe estar un trabajo para poder ser ejecutado.

Walltime	• tiempo de ejecución de un trabajo.

Operativa de envío de trabajos

El usuario se conecta con el servidor

- habitualmente de forma remota

Los usuarios envían sus trabajos al servidor mediante scripts

- qsub<script >

El script está formado por:

- Especificación de shell (opcional).
- Directivas PBS: empeladas para solicitar recursos (CPUs, memoria, licencias) o especificar atributos (walltime).
- Tareas: programas o comandos.

El sistema devuelve un identificador de trabajo para posteriores acciones sobre él

- monitorización, borrado, etc.

Ejemplo de script

- #!/bin/sh
- #PBS -l walltime=1:00:00
- #PBS -l mem=400mb,ncpus=4
- ./my_application

Referencias

- Sistemas Operativos Modernos. Andrew S. Tanenbaum. Pearson Education, 2009.
- http://www.ibm.com/developerworks/linux/library/
- D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, “Theory and practice in parallel job scheduling,” in IPPS ’97:Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing. London, UK: Springer, 1997, pp. 1–34.
- D. Feitelson, L. Rudolph, and U. Schwiegelshohn, “Parallel jobscheduling–a status report,” Lecture Notes in Computer Science, vol.3277, pp. 1–16, 2005.
- PBS Professional User’s Guide, Altair PBS Professional 10.2
- http://www.youtube.com/watch?v=0G0z8SauSDY#t=55