Migración de procesos a nivel usuario[1]

Juan Luis Chaves - Leandro León
(juanluis|lrleon)@cemisid.ing.ula.ve
Centro de Estudios en Microelectrónica y Sistemas Distribuidos (CEMISID)
Centro Nacional de Cálculo Científico (CECALCULA)
Universidad de Los Andes - Mérida - Venezuela

Resumen

La migración es una abstracción inherente a la distribución. Empero, ninguna solución propuesta ha sido satisfactoria. Este trabajo describe un mecanismo de migración de procesos, enteramente implantado a nivel usuario, que selecciona y sincroniza dinámicamente la transferencia. El usuario puede especificar el orden de transferencia de los datos, los tiempos en que las transferencias deben ser realizadas, cuáles datos deben ser transferidos y el tamaño de los datos a transferir; estos parámetros pueden ser ajustados dinámicamente, inclusive, durante el transcurso de la migración. Mediciones reportan una latencia de inactividad de 12,3 mili-segundos (Ethernet 10 Mb/sec), la más baja hasta el presente reportada. El mecanismo es portátil y no sacrifica ninguna funcionalidad de un sistema distribuido. Aplicaciones que no requieren migrar no pagan ningún coste por la presencia de la migración.

Palabras clave: Sistemas Distribuidos, Migración de Procesos, Equilibrio de Carga

1 Introducción

La migración es el acto de mover un proceso en ejecución desde un sitio a otro. La migración es motivada principalmente por el equilibrio de carga, la tolerancia a fallas, la administración del sistema y el mejoramiento de la localidad de acceso a datos remotos.

A tenor de la migración, una gran cantidad de investigaciones han sido realizadas y algunos resultados importantes han sido reportados. A pesar del enorme interés desde la misma aparición de los sistemas distribuidos, la migración casi no ha salido del contexto universitario y ningún intento comercial serio ha sido emprendido. Este artículo intenta atacar los problemas que han impedido que la migración sea un soporte popular en todo sistema distribuido.

1.1 El problema

En este artículo, nosotros argüimos que la migración aún no ha sido adoptada como mecanismo de facto en cualquier sistema distribuido por las siguientes razones:

- *Poca portatibilidad:* Los mecanismos de migración reportados han sido diseñados para plataformas materiales (hardware) y/o sistemas operativos específicos.
- *Pobre desempeño:* Pocos resultados en la migración han obtenido rendimientos atractivos para ser usados por políticas de equilibrio de carga.
- *Impacto de la migración sobre el desempeño global:* Muchos esquemas de migración añaden carga adicional al sistema. En muchos casos, aplicaciones que no requieren la migración son penalizadas por su presencia en el sistema.
- *Poca adaptación:* La mayoría de las estrategias de migración no son adaptables, ni a la gama de aplicaciones posibles, ni a los cambios ambientales que transcurran durante la migración. El diseñador de una política de equilibrio de carga considera el costo del mecanismo de migración como un parámetro en lugar de considerarlo como una variable dinámicamente modificable. Una política de equilibrio no puede prepararse a cambios súbitos de carga durante la migración.
- *Perturbación de la migración sobre el equilibrio de carga:* En muchos casos, la migración impone una carga adicional excesiva sobre el sistema que paradójicamente incide sobre la carga global.

[1]Esta investigación fue financiada por el CDCHT-ULA bajo el proyecto No. I-620-98-02-AA

Es muy difícil diseñar un algoritmo general de migración que sea adecuado a todos los tipos de aplicación. El diseñador del mecanismo de migración no puede conocer todos los tipos posibles de aplicación. Como consecuencia, los mecanismos de migración son diseñados para aplicaciones específicas. Un sistema operativo de uso general no puede, entonces, ofrecer una migración universal adecuada a cualquier tipo de aplicación.

1.2 Los objetivos de este trabajo

En este artículo, nosotros describiremos una interfaz usuario y una implantación que, combinadas, proporcionan un sistema con las siguientes características:

- *Funcionalidad:* Nuestro sistema puede utilizar cualquier estrategia de transferencia existente.
- *Adaptabilidad:* Nuestro sistema puede modificar dinámicamente el mecanismo de migración, aun durante el proceso mismo de migración.
- *Desempeño:* Nuestro sistema exhibe un desempeño superior al mejor sistema reportado.
- *Flexibilidad:* La parte usuario de nuestro sistema está estructurada de manera tal que el usuario pueda aportar al sistema conocimiento acerca el tipo de aplicación. Adicionalmente, el sistema permite que el usuario pueda configurar la aplicación y ejecutar el mecanismo de migración que más le convenga.
- *Portatibilidad:* Nuestro sistema es naturalmente portátil hacia cualquier plataforma material y sistema operativo.
- *Heterogeneidad:* Nuestro enfoque permitiría, bajo condiciones especiales, la migración sobre material heterogéneo. Migración sobre el mismo material, pero bajo diferentes sistemas operativos, es completamente posible.

1.3 La propuesta

En este trabajo nosotros argüimos que los objetivos expuestos pueden ser alcanzados si una gran parte de la migración es implantada a nivel usuario. Nuestro enfoque se enmarca en el principio extremo a extremo [17] y consiste en permitir que el usuario aporte el conocimiento que no puede disponer, ni el implantador del mecanismo de migración, ni el implantador de la política de equilibrio de carga. Esta funcionalidad se logra mediante una biblioteca básica de migración, accesible al usuario, que le permita aportar el conocimiento en cuestión.

La biblioteca de migración es encadenada sólo a las aplicaciones que requieran migrar. Aplicaciones que no migren, no pagan ningún coste por el soporte de migración.

Aplicaciones que deseen migrar sin aportar conocimiento, utilizan una biblioteca de migración por omisión. Existirían otras bibliotecas de migración especializadas según el tipo de aplicación.

La interfaz permite al usuario especificar puntos donde la migración no sería deseable. Del mismo modo, el usuario también puede especificar el orden de transferencia de los datos, los tiempos en que las transferencias deben ser realizadas, cuáles datos deben ser transferidos y el tamaño de los datos a transferir. Sólo el escritor de la aplicación está en capacidad de ofrecer al sistema todo este conocimiento.

Toda acción efectuada por el usuario a través de la biblioteca de migración es registrada por el sistema. El sistema ofrece otra interfaz que permite intervenir externamente una migración en curso. De esta manera, una política de equilibrio de carga puede disponer de esta información y considerarla en sus decisiones de equilibrio. Por ejemplo, la política de equilibrio podría "retroceder" la migración si ésta se percata de que la migración ya no es necesaria porque las condiciones de carga han cambiado.

Todo el sistema es implantado a nivel usuario, sin ninguna modificación del núcleo del sistema operativo. Esta característica le proporciona portatibilidad al mecanismo e independencia del sistema operativo.

La transferencia de datos es realizada por un ejecutivo contenido en la biblioteca de migración. Esta biblioteca podría ser modificada para efectuar las conversiones necesarias en plataformas materiales heterogéneas.

Las medidas de rendimiento, mostradas en la sección § 4, demuestran la viabilidad de nuestra propuesta.

2 Los problemas de la migración y sus antecedentes

En general, un servicio de migración debe lidiar con los siguientes problemas:

Congelamiento: El congelamiento consiste en la detención de la ejecución del proceso y en la extracción de su estado de ejecución. El mecanismo de extracción puede variar a través de diferentes sistemas operativos y plataformas materiales (hardware). En consecuencia, el congelamiento es la parte menos portátil de la migración.

Transferencia: La transferencia es el acto de copiar todo el estado del proceso desde el sitio origen hacia el sitio destino de la migración.

Actualización de los lazos de comunicación: El proceso migrante puede enviar y recibir mensajes. La actualización consiste, entonces, en actualizar los lazos de comunicación de aquellos procesos que envíen mensajes hacia el proceso migrante.

Protocolo: Todas las acciones anteriores deben sincronizarse de manera coherente. El sitio destino debe poseer recursos suficientes para asegurar el arribo del proceso migrante. La transferencia debe realizarse después del congelamiento. La reanudación de la ejecución en el sitio destino debe efectuarse luego de la transferencia. Estas acciones deben ser sincronizadas y supervisadas por un protocolo entre los sitios involucrados en la migración.

Durante el tiempo en que la ejecución es congelada, el proceso migrante no efectúa ningún progreso. Esta "latencia de inactividad" constituye el principal problema de la migración por las siguientes razones:

1 En un contexto de equilibrio de carga, es razonable pensar que se acelerará la ejecución del proceso migrante. En este sentido, el tiempo de inactividad podría actuar en detrimento del tiempo de finalización del proceso.

2 Políticas de equilibrio de carga seleccionan procesos pesados para ser migrados. Empero, mientras más pesado sea el proceso, más costosa es su migración y más largo es el tiempo de inactividad.

3 La latencia inactividad puede extenderse hacia aquellos procesos que mantengan comunicación con el proceso migrante, pues, durante el tiempo de inactividad, el proceso migrante no puede procesar mensajes dirigidos hacia él.

2.1 Protocolo de migración

La mayoría de los sistemas comienzan la migración con una fase de "negociación" con el sitio destino que asegure que él dispone de los recursos necesarios para albergar al proceso migrante. Luego, se debe señalar de alguna forma que el proceso migrante debe ser congelado. El congelamiento debe ser notificado de manera tal que se pueda iniciar la transferencia. De igual modo, la finalización de la transferencia debe ser notificada de manera tal que se pueda reanudar la ejecución en el sitio destino. Finalmente, los recursos del proceso migrante deben ser liberados del sitio origen.

DEMOS/MP [13] y Charlotte [4] están entre los pocos sistemas que han reportado el protocolo de migración. Ambos sistemas han seguido la secuencia de eventos descrita en el párrafo anterior.

2.2 Estrategias de transferencia

Puesto que la transferencia es la fase dominante en la latencia de inactividad, la mayoría de las investigaciones se han concentrado en realizar la transferencia en el menor tiempo posible.

Algunos sistemas, por ejemplo, DEMOS/MP [13] y Charlotte [4], han implantado la "copia total" en la cual el proceso migrante no es reanudado hasta que todo el estado del proceso ha sido transferido. Este es el tipo de transferencia más costoso en tiempo de inactividad.

Theimer et al [20] concibieron una estrategia de dos fases. La primera es llamada "pre-copia" y consiste en transferir todo el espacio de direccionamiento. Luego, se suspende el proceso y sólo se transfieren las partes del proceso modificadas durante la pre-copia. La pre-copia causa más carga que la copia total y el tamaño de los datos modificados crece conforme aumenta el tamaño del proceso a migrar.

Zayas [21] identificó la fase de transferencia como el "cuello de botella" de la migración. Zayas

propuso transferir la información mínima para reanudar la ejecución en el sitio destino. El resto del contenido del proceso es copiado "a la demanda", cuando se efectúa la primera referencia. La estrategia de Zayas es llamada "copia por referencia". Sobre un conjunto representativo de 7 aplicaciones, Zayas reportó que el promedio de bytes intercambiados entre sitios disminuía en un 58,2 % y la cantidad de mensajes disminuía en un 47.8 %. Empero, la copia por referencia exhibe el problema de "dependencias residuales" [7]. Los datos no referenciados deben ser mantenidos indefinidamente en el sitio origen. La carga aumenta significativamente conforme aumenta la cadena de migraciones sucesivas. En añadidura, la caída de un nodo rompe la cadena.

Douglis y Ousterhout [7] propusieron una variación del esquema de Zayas en la cual, en paralelo con la reanudación de la ejecución, el espacio de direccionamiento es "vaciado" hacia un sistema de archivos. Esta estrategia delega las dependencias residuales y la sensibilidad a fallas al sistema de archivo.

Milojicic et al [12] modificaron el micro-núcleo Mach e implantaron una migración que permite seleccionar la estrategia de transferencia. Todas las estrategias anteriores son incluidas e, inclusive, una nueva estrategia, denominada "lectura avanzada", fue propuesta e implantada. Esta última estrategia fue formalizada en [14] y fue denominada "post-copia". La estrategia consiste en copiar el estado mínimo para reanudar en el destino y luego copiar de fondo (background) el resto del estado. El objetivo de Milojicic et al fue paliar la rigidez impuesta por la estrategia de transferencia y poder especializarla al tipo de aplicación. Sin embargo, desde la perspectiva del sistema operativo, es imposible conocer la estrategia de transferencia más adecuada para un proceso.

Roush [15] implantó sobre el sistema Choices una migración de alto desempeño que él denominó "libre de congelamiento" a causa de su muy baja latencia de inactividad (13,9 mili-segundos). Su mecanismo está basado en el uso de un protocolo de ráfaga (blast).

Todos los trabajos presentados en esta sección fueron hitos en la migración. A la excepción del trabajo de Milojicic et al [12], todos los trabajos sufren el problema de inadaptación a los diversos tipos de aplicaciones. Empero, Milojicic et al no consideran ningún tipo de información que pueda aportar el escritor de la aplicación a migrar.

2.3 Actualización de lazos de comunicación

Un enfoque cándido para actualizar lazos de comunicación es la difusión (broadcasting) de la nueva localidad hacia los procesos ligados al proceso migrante. Este método sólo es considerado para sistemas de poca escala, pues la difusión produce considerables efectos adversos en una red. Una variante de difusión fue utilizada en el sistema Charlotte [4].

Otro enfoque cándido es denominado "prueba y actualización" (test and update). Antes de que se efectúe una migración, el cliente verifica si el servidor realmente se encuentra en la localidad esperada. Si el servidor está ausente, el cliente pregunta por la nueva localidad a un servicio especial de localización. Este enfoque fue utilizado por los sistemas Emerald [9], SOS [18] y Amber [6] para determinar si un lazo era local o no.

El enfoque de actualización más popular es denominado "lazo de persecución" (forward address). Un lazo de persecución es una indicación dejada en el sitio origen de la migración y que apunta al sitio destino. Cuando un cliente envía un mensaje, éste llega normalmente al sitio origen. Entonces, desde allí es enviado al sitio destino. Las cadenas de lazos se incrementan según el número de sitios nuevos que se van visitando en migraciones sucesivas. Esto recrea el problema de dependencias residuales [7]. Aparte de la degradación de desempeño y sensibilidad a fallas, los lazos de persecución también presentan el problema de que su gestión debe ser incorporada al protocolo de migración [11], pues los enlaces deben ser procesados antes de reanudar la ejecución en el sitio destino. En añadidura, los lazos de persecución requieren un mecanismo de recolección de lazos no utilizados (garbage collector). Esto hace la instrumentación del sistema más compleja y afecta el rendimiento global del sistema distribuido. Los lazos de persecución han sido utilizados en los sistemas Emerald [9] y Demos/MP [13].

Cualquier forma de lazo de persecución tiene el problema de que la reanudación no puede realizarse hasta que el lazo esté conformado, lo que incide sobre la latencia de inactividad.

El último enfoque de actualización es denominado "recuperación de mensajes perdidos". En

este enfoque, un mensaje enviado a un proceso falla si éste ha migrado. Se inicia, entonces, un mecanismo de recuperación del mensaje perdido que consiste en hacer llegar el mensaje a la nueva localidad. Esta técnica es sencilla de implementar y no interfiere con el protocolo de migración [11]. Las dependencias residuales son suprimidas. Empero, es más difícil de adaptar a los sistemas de gran escala. Una variante de recuperación de mensajes perdidos fue utilizada en el sistema Amoeba [19].

3 Soporte usuario para una migración eficiente

En esta sección presentamos una propuesta que ataca los problemas descritos y que está basada en los lineamientos presentados en las secciones subsiguientes.

3.1 Reducción de los datos a transferir

La reducción de los datos a transferir es lograda mediante una división funcional del espacio de direccionamiento en zonas según el tipo de utilización. La figura 1 ilustra una división utilizada en un prototipo implantado por [11] sobre el micro-núcleo Chorus [16]. La condición esencial de esta división es que si el proceso migra, todas estas zonas se copiarán exactamente en las mismas direcciones virtuales.

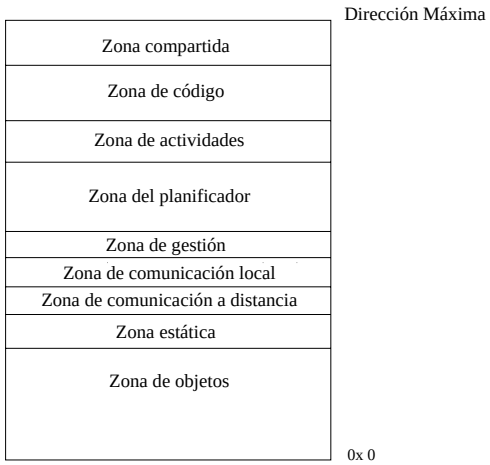

Figura 1: División del espacio de direccionamiento de un proceso

La zona compartida contiene el código correspondiente a símbolos dinámicamente resueltos mediante librerías compartidas. La zona de código contiene el código estático de la aplicación. La zona estática contiene datos estáticos de la aplicación apartados por el compilador.

El resto de las zonas son dinámicamente gestionadas. Sus funciones son descritas a continuación:

- *Comunicación local:* Donde se colocan las estructuras de datos relacionadas con la comunicación entre procesos locales.
- *Comunicación remota:* Análoga a la anterior pero para la comunicación remota.
- *Zona de objetos:* Esta es la zona destinada a los datos dinámicos requeridos por la aplicación.
- *Zona de actividades (threads):* Esta zona alberga las pilas de ejecución de las actividades del proceso.
- *Zona de gestión:* Esta zona mantiene las estructuras de datos dinámicas que pertenecen a la gestión del proceso y que no son parte de la aplicación. Por ejemplo, estados de control de código almacenado en bibliotecas compartidas.
- *Zona de planificación:* Esta zona mantiene las estructuras de datos y el estado del planificador de actividades. Esta zona sólo es pertinente para procesos multi-actividad.

Tradicionalmente, las funcionalidades de estas zonas son gestionadas en una sola zona comúnmente llamada montículo (heap). En consecuencia, la memoria dinámica contiene objetos pertenecientes a la aplicación y objetos pertenecientes al sistema que no son manejados por

la aplicación. En esta clase de sistemas, es imposible distinguir cuáles objetos pertenecen a la aplicación y cuáles al sistema. La separación funcional en zonas permite, entonces, aislar los datos pertenecientes a la aplicación en una zona única: la zona de objetos.

Todas las zonas dinámicas son gestionadas por manejadores de memoria independientes. Esto conlleva dos ventajas. La primera es la reducción de la contención de gestión de memoria, pues los manejadores son protegidos por cerrojos (locks) diferentes. La segunda es la posibilidad de especializar el manejador según la función de la zona. Por ejemplo, puesto que las pilas de las actividades siempre son múltiplos del tamaño de página virtual del sistema, el correspondiente manejador de memoria puede ser especializado para bloques múltiplos de página.

La reducción de la transferencia consiste, entonces, en identificar cuáles zonas no requieren ser copiadas en el sitio destino, cuáles zonas son pre-copiables, cuáles zonas conforman el estado mínimo de ejecución y cuáles zonas son copiables por referencia. La tabla 1 indica esta clasificación.

Clasificación	**Zonas**
Descartable (no se requiere copiar)	Zona compartida, zonas de comunicación
Estado mínimo	Zona de planificación
Pre-copiable	Zona código, Partes constantes de las zonas estática y de objetos
Referenciable	Todas las zonas excepto las desechables

Tabla 1: Clasificación de zonas memoria respecto al tipo de copia

Las zonas de comunicación pueden descartarse si se garantiza que el congelamiento no ocurrirá durante el procesamiento de un mensaje recibido. Esto es posible porque la planificación usuario toma en cuenta a la comunicación (ver § 3.2).

Las bibliotecas compartidas pueden desecharse completamente si cada sitio puede accederlas. En este caso, se puede disminuir substancialmente la cantidad de datos a transferir mediante el uso intensivo de bibliotecas compartidas. Las bibliotecas compartidas siempre son mapeadas en la zona compartida, lo que facilita la manipulación de los símbolos ya resueltos.

El estado mínimo de ejecución es el estado del planificador. Así pues, esta es la única zona que se requiere copiar para poder reanudar la ejecución. El resto de los datos pueden perfectamente ser copiados por referencia.

Las pilas de las actividades pueden ser copiadas directamente o por referencia. Note que es posible conocer la próxima pila de ejecución, pues el planificador es a nivel usuario.

Una eventual objeción a dividir el espacio de direccionamiento es la limitación que se impone sobre el máximo tamaño de datos que puede ser direccionado por el proceso. Esta limitación puede ser severa en arquitecturas de 32 bits que ofrecen una longitud máxima de 4 Gbytes. En este tipo de arquitecturas, el espacio puede ser aprovechado mediante un estudio cuidadoso del uso de memoria de la aplicación. Este estudio podría ser automatizado entre el compilador y el encadenador. Sin embargo, esta necesidad devendría obsoleta, pues las nuevas arquitecturas utilizan espacios de direccionamiento de 64 o más bits. Chase et al [5] demostraron que un sólo espacio de direccionamiento de 64 bits era suficiente para contener aplicaciones grandes, reales y complejas. El estudio de Chase et al fue para todos los procesos del sistema. Nuestro enfoque limitaría a 64 bits cada proceso del sistema.

3.2 Planificación usuario

El congelamiento es llevado a cabo a través de un planificador de actividades a nivel usuario. El planificador exporta primitivas para la migración. La tabla 2 muestra cada primitiva y su función.

Cada vez que ocurre un cambio de contexto (context switch) entre dos actividades, el estado de la actividad en ejecución es guardado en una estructura de dato tradicionalmente conocida como "bloque de control de actividad" (thread control block). El planificador mantiene, entonces, el estado de todas las actividades del proceso.

Todas las estructuras de datos del planificador son guardadas en la zona de planificación. De este modo, la migración consulta la dirección base del planificador y su longitud. Esta región es copiada en el destino exactamente en las mismas direcciones de memoria.

Primitiva	Descripción
`void sq_suspend()`	Suspende toda ejecución
`void sq_resume()`	Reanuda ejecución del planificador
`void *sq_base_addr()`	Dirección base del planificador
`size_t sq_size()`	Tamaño total de las estructuras planificador

Tabla 2: Primitivas de planificación para la migración

Las actividades usuario han sido cuestionadas por diversos problemas de consistencia. Estos problemas pueden ser resueltos mediante un enfoque de planificación híbrida entre el proceso y el núcleo del sistema operativo denominado "activaciones del planificador" [2]. Este enfoque es perfectamente compatible con las necesidades de nuestro planificador.

Aplicaciones que no deseen migrar, pueden utilizar cualquier otro planificador. Aplicaciones mono-actividad son mucho más fáciles de congelar a través de señales.

3.3 Sincronización a nivel usuario

El escritor de la aplicación puede especificar diversas acciones a ejecutar en el transcurso de los eventos relacionados con la migración. Los eventos señalados al proceso son: el inicio de la migración, el inicio del congelamiento, el congelamiento del proceso y la reanudación de la ejecución en el sitio destino. Para señalizar estos eventos, el sistema efectúa llamadas a las funciones apuntadas por los punteros indicados en la tabla 3. El sistema maneja 3 modos de sincronización: pre-copia, copia, y post-copia.

Estado	Nombre del Apuntador	Descripción
Pre-copia	`void (*pre_copy_fct)()`	Llamada al inicio de la migración
Copia	`void (*copy_fct)()`	Llamada cuando el proceso es congelado
Post-copia	`void (*post_copy_fct)()`	Llamada luego de la reanudación
Reanudación	`bool target_resume()`	Reanuda ejecución en destino
Aborto	`void migration_abort()`	Aborta la migración
Suspensión	`void suspend_migration()`	Suspende migración
Reanudación	`void resume_migration()`	Reanuda migración

Tabla 3: Primitivas de sincronización

El usuario puede escribir y sobrecargar las primitivas llamadas mediante apuntadores a funciones.

El modo pre-copia es activado al comienzo de la migración. `(*pre_copy_fct)()` es llamada al inicio de este modo. Por omisión, esta función copia toda la zona de código. Al término de esta función, el sistema entra en modo copia e inicia el congelamiento del proceso. `(*copy_fct)()` es llamado cuando el proceso es congelado. Por omisión, esta función copia toda la zona del planificador, toda la zona de objetos, toda la zona de gestión y la primera página de la pila correspondiente a la primera actividad en la cola del planificador. La ejecución en el destino es automaticamente reanudada al término de `(*copy_fct)()`. En este momento, el sistema entra a modo post-copia y la función `(*post_copy_fct)()` es ejecutada. Por omisión, esta función copia el resto del espacio de direccionamiento al sitio destino. Al término de la post-copia, el sistema libera los recursos en el sitio origen, aun si el espacio de direccionamiento no ha sido totalmente copiado.

`target_resume()` permite cambiar explícitamente de modo copia a post-copia. Si esta función es llamada, entonces el sistema hace el mejor esfuerzo por reanudar la ejecución en el destino.

La migración puede ser explícitamente abortada (`migration_abort()`), o suspendida (`suspend_migration()`) para luego ser reanudada (`resume_migration()`). Las dos últimas fun-

Nombre de la primitiva	Descripción
`size_t size_obj_zone(Zone_t);`	Devuelve el tamaño de la zona. `Zone_t` puede referir a las zonas de objetos, de código o la estática
`void* base_addr_zone(Zone_t);`	Devuelve la dirección base de la zona.
`size_t size_zone(Zone_t);`	Devuelve el tamaño de la región cero de la zona
`size_t copy_out(Zone_t, void *addr, size_t size)`	Transfiere `size` bytes a partir de `addr`

Tabla 4: Primitivas de soporte para la copia

ciones aceptan como parámetro un puntero a una función. `suspend_migration()` llama a la función antes de efectuar la suspensión. `resume_migration()` llama a la función antes de efectuar la reanudación.

Mediante esta interfaz, el usuario puede especificar su propia estratégia de migración. En cada uno de los modos, el usuario puede consultar el estado del sistema y modificar dinámicamente la migración.

El modo actual de migración y cuáles y cuántas páginas han sido copiadas pueden ser consultados por entes externos. Esta funcionalidad permitiría a una política de equilibrio intervenir y modificar el proceso de migración.

La tabla 4 ilustra algunas de las primitivas que dispone el usuario para efectuar la copia. La más importante es `copy_out()`, cual puede ser llamada bajo cualquier modo y efectúa la copia explícita de los datos.

3.4 Copia por referencia

A partir del modo post-copia, el sistema activa la copia por referencia de páginas referenciadas en el destino que aún no han sido copiadas.

Zayas [21] demostró la importancia de la copia por referencia. Aunque ella no es implantada por los sistemas operativos tradicionales, tales como UNIX y Windows-NT, ella es perfectamente implantable a nivel usuario mediante el mecanismo de manejo de señales.

3.5 Actualización por fracaso de envío de mensaje

Dos aspectos son esenciales para minimizar el impacto de la actualización de los enlaces de comunicación en la migración. Primero, la actualización no debe retardar la reanudación de la ejecución. Segundo, se debe reducir el tráfico de mensajes. En esta sección nosotros proponemos una variante denominada "actualización por fracaso de envío de mensaje" que satisface estos requerimientos. La técnica consiste en los siguientes lineamientos:

1 El sistema de comunicación debe identificar los destinatarios de mensajes mediante "puertos" opacos, únicos en el tiempo, que oculten las coordenadas físicas del destinatario. Esta abstracción es exportada por muchos sistemas operativos distribuidos, Chorus [16], por ejemplo, y es fácilmente simulable en sistemas operativos tradicionales.

2 Cuando se inicia la migración, se construye un nuevo puerto por el cual el destinatario recibirá los mensajes. De esta manera, el puerto en el sitio origen es "invalidado". El nuevo puerto es asincrónicamente enviado a un servicio distribuido de localización. Por limitaciones de espacio, este servicio no será discutido. Implantaciones de éste servicio pueden ser revisadas en [11, 3].

3 Mensajes que arriban al origen durante la pre-copia son aceptados y procesados normalmente. Mensajes que arriban durante la copia son retenidos hasta que la ejecución es reanudada. Cuando la ejecución es reanudada, estos mensajes retenidos son rechazados, pero la nueva localidad, expresada en el nuevo puerto, es enviada con el rechazo. De esta manera, el remitente se percata de que debe reenviar el mensaje al nuevo puerto.

4 Cuando la migración ha finalizado, el envío de mensajes hacia el origen fracasará por inexistencia de puerto. La verificación del puerto es la primera acción que ejecuta un sistema de comunicación al arribo de un mensaje. Consecuentemente, el remitente se percata rápidamente de la excepción. La excepción es el evento que le indica al remitente que el envío ha fracasado, posiblemente porque el destinatario ha migrado. Así pues, el remitente emite una solicitud al sistema de localización. El sistema de localización encuentra el nuevo puerto y se lo retorna al remitente quien reenvía el mensaje. Esquemas distribuidos altamente eficientes para efectuar la localización pueden ser encontrados en [11, 3].

La virtud de esta técnica es que ella es perezosa: sólo se actualizan los lazos que se utilizan. A condición de que la localización sea eficiente, nuestro enfoque es menos costoso que los lazos de persecución, pues sólo requiere 5 mensajes, cantidad independiente del número de migraciones.

Esta técnica es perfectamente implantable a nivel usuario.

3.6 Protocolo de migración

Cuando se efectúa la migración, una imagen vacía del proceso es creada en el sitio destino. Esta imagen albergará el proceso migrante y será denominado "proceso destino" durante el resto del discurso. Inicialmente, el proceso destino carga un ejecutivo de gestión encargado del protocolo del lado destino. La parte origen del protocolo está presente en una biblioteca compartida cargada en el proceso migrante, cual será denominado "proceso origen". La figura 2 ilustra la secuencia del protocolo. El protocolo sólo involucra, entonces, al proceso origen y al proceso destino. Esta característica tiene la bondad de permitir múltiples migraciones.

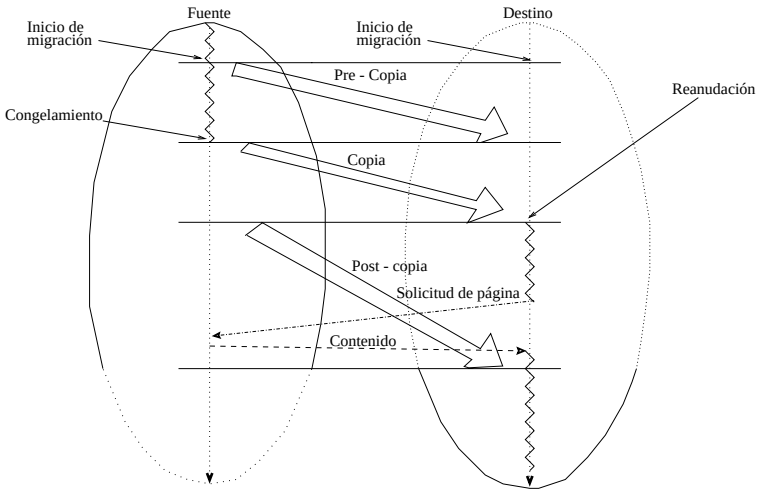

Figura 2: Protocolo de migración

La complejidad y rapidez del protocolo se reducen substancialmente si se elimina la fase de negociación. Esto puede parecer peligroso porque el sitio destino puede carecer de recursos suficientes para albergar el proceso migrante. Empero, una política correcta de equilibrio de carga seleccionará como destino un sitio con suficientes recursos. Así pues, nuestro protocolo no tiene negociación con el sitio destino y asume de facto que la migración será posible. Es, pues, responsabilidad del ente que ordene la migración, asegurar que el sitio destino disponga de los recursos necesarios.

3.7 Implantación

Nosotros hemos implantado un prototipo alfa que valida nuestras ideas sobre el sistema operativo distribuido Chorus [16]. El prototipo fue denominado COOL-LDM [10, 11] y es un descendiente del sistema distribuido a objetos COOL2 [1], también implantado sobre Chorus. COOL-LDM está disponible para verificación bajo posesión legal de licencia del micro-núcleo Chorus y autorización expresa de la compañía *Chorus Systems*.

COOL-LDM implanta todas las ideas propuestas en esta sección a la excepción del planificador. En su lugar, el estado de las actividades es extraído por modificación del micro-núcleo.

Una implantación más madura, que será de distribución pública, está siendo desarrollada en el CEMISID-ULA sobre el sistema operativo Linux. El prototipo está desarrollado en un 40%: el sistema IPC ya está en versión beta y es libremente distribuido vía ftp anónimo en `ftp.cemisid.ing.ula.ve`, directorio `/pub/aleph/ipc`. La copia por referencia y el planificador de actividades están en versión alfa y estarán disponibles prontamente. Nosotros estimamos que una versión alfa completa del sistema estará disponible al momento de realización de esta conferencia.

4 Desempeño

Las mediciones que reportaremos fueron realizadas sobre el prototipo COOL-LDM. Los experimentos fueron realizados sobre una red, aislada, Ethernet de 10 Mbytes/sec con tarjetas de red `3com 3c501`, de buses internos de 8 bits. Se utilizaron dos máquinas modelos *Compaq Deskpro 386/25e* con procesadores Intel 386, una frecuencia de reloj de 25 Mhz, 16 Mbytes de memoria y 32 Kbytes de cache. El sistema operativo utilizado fue el micro-núcleo `Chorus r2.3` versión de núcleo 3.5. El compilador `C++` utilizado fue el *ATT-cfront* versión 3.0.3, cual genera código cruzado hacia lenguaje `C`. Las salidas `C` de `cfront` fueron compiladas con el compilador `gcc -O3` versión 2.6.3.

Sobre esta plataforma, se ejecutó un proceso con 8 actividades donde la función `(*copy_fct)()` (§ 3.3) fue sobrecargada para sólo copiar la zona del planificador. En este experimento se obtuvo una latencia de inactividad de 12,3 mili-segundos. Esta es la latencia de inactividad más baja hasta el presente reportada.

La latencia de inactividad no es suficiente para evaluar el desempeño de la migración. Para complementar, se realizaron mediciones de la latencia de la función `copy_out()` (§ 3.3). La tabla 5 ilustra las latencias para varios tamaños de copia.

Tamaño (Kbytes)	Tiempo en segundos
16	0.139
32	0.243
64	0.458
128	0.913
256	1.821
512	3.659
1024	7.356
2048	14.770
4096	29.387
9192	59.500

Tabla 5: Latencia de `copy_out()`

La importancia de estas medidas es que son muy próximas al mínimo impuesto por el material y por el sobrecosto (overhead) del sistema operativo.

El servicio de localización distribuido de COOL-LDM ofrece garantías probabilísticas para encontrar un puerto creado recientemente en aproximadamente 0,79 mili-segundos. Con esta latencia, un mensaje enviado a un proceso recientemente migrado toma 39,4 mili-segundos. Si el mensaje es enviado mucho después de la migración, encontrar el nuevo puerto puede tomar 21,25 mili-segundos. En este caso, el envío de un mensaje a un proceso migrado toma 68,7 mili-segundos.

5 Perspectivas y futuro trabajo

Aunque prometedores, los desarrollos y resultados presentados en este artículo deben ser sometidos a más verificaciones. Uno de los atractivos de nuestro enfoque es que una política de equilibrio de carga podría considerar el costo de la migración como una variable dinámicamente modificable. En este sentido, nosotros consideramos los puntos siguientes como los más importantes a investigar:

- ¿Cuáles páginas del espacio de direccionamiento deben ser seleccionadas para enviar en precopia, copia y post-copia? Este problema debe ser abordado en relación al tipo de proceso, la información proporcionada por el usuario y el estado actual de carga del sistema.

- ¿De qué manera la transferencia debe reaccionar ante variaciones del ambiente de ejecución durante la migración?
- ¿Cómo caracterizar aplicaciones en función del mecanismo adecuado de migración? Esta caracterización permitiría construir bibliotecas de migración especializadas según el tipo de aplicación.

A nivel del mecanismo, resta trabajo por realizar. Sería interesante investigar acerca protocolos especializados para la transferencia masiva de datos. COOL-LDM utilizó el protocolo IPC de Chorus, cual es orientado a cliente-servidor. Nuestro nuevo prototipo utiliza directamente una conexión TCP/IP. En este sentido, un futuro desarrollo sería la construcción de un protocolo de ráfaga tal como el señalado en § 2.2 y utilizado en [15].

Otro desarrollo importante sería incorporar la zona de planificación en la función `copy_out()`. Esto no se ha realizado por el peligro de que el usuario corrompa el planificador. Esta posibilidad es muy importante porque el usuario podría conocer y copiar la próxima actividad que será ejecutada cuando la ejecución sea reanudada en el sitio destino.

Aplicaciones orientadas a objetos suelen delegar la liberación de memoria a un mecanismo de recolección de basura (garbage collector). Puesto que los objetos están en una zona específica, el mecanismo recolector podría compactar la zona de objetos antes de efectuar su transferencia. Esto podría reducir dramáticamente el tamaño de la zona de objetos y acelerar la migración.

Finalmente, un aspecto no menos importante lo constituye la migración en ambientes heterogéneos. En este sentido, el CEMISID-ULA está iniciando investigaciones en contextos CORBA [8] donde los objetos permitirían relocalización en direcciones diferentes entre zonas de objetos heterogéneas. Puesto que la transferencia es realizada a nivel usuario, algunas de las bibliotecas en los sitios involucrados podrían efectuar las conversiones necesarias. Recientemente, el CEMISID-ULA ha emprendido investigaciones para instrumentar este tipo de migración en JAVA.

6 Conclusión

Este trabajo aún no posee la madurez suficiente para enunciar conclusiones contundentes. Por esa razón, nosotros nos limitaremos a expresar nuestras intuiciones acerca las bondades de nuestro trabajo.

Nosotros pensamos que la migración es un problema donde el argumento extremo a extremo [17] es completamente aplicable. La estrategia de migración ideal es aquella que anticipa y copia las páginas exactamente en el orden en que serán referenciadas. Cuáles y cuántas páginas copiar y cuándo copiarlas son tres preguntas completamente dependientes de la aplicación. Sólo la aplicación es capaz de responder estas preguntas. Así pues, sólo la aplicación puede contribuir decisivamente en una transferencia adecuada.

La planificación no es una razón que justifique implantar la migración a nivel sistema. Como se describió en § 3.2, el estado de ejecución puede ser manejado a nivel usuario. Del mismo modo, la actualización de lazos de comunicación tampoco justifica una migración sistema. De hecho, el fracaso de envío de mensaje es implantado enteramente en biblioteca.

En nuestras experiencias, el esquema de migración que proponemos es implantable en cualquier sistema operativo moderno. En el contexto del prototipo que estamos desarrollando en Linux, la única parte dependiente de plataforma es del orden de decenas de líneas de código correspondientes al cambio de contexto de las actividades. Así pues, intuitivamente, creemos que nuestro sistema será portátil de plataforma en plataforma y de UNIX en UNIX.

Nosotros concluimos este artículo con una palabra de prudencia. La mayoría de las investigaciones en migración han reivindicado una separación completa entre el mecanismo y la política. La ausencia de un paradigma en migración sugiere que esta separación aún no ha sido alcanzada. En cierta medida, esta separación es utópica porque el diseño de una política puede depender del mecanismo seleccionado. Determinar si el mecanismo que proponemos está separado de la política sería el objeto de futuras investigaciones.

7 Agradecimientos

Philippe GAUTRON y Christian JACQUEMOT de *Chorus Systems*, así como Eric GRESSIER del CNAM, Francia, contribuyeron en la realización del prototipo COOL-LDM. Carlos Nava contribuyó

decisivamente en la elaboración del nuevo prototipo. Andrés Arcia contribuyó en la elaboración del sistema de localización. Gilberto Díaz contribuyó decisivamente en la diagramación de éste artículo.

Referencias

[1] Amaral, P. *PAS, a framework for stuying the implementation of multiple uniform address spaces.* PhD thesis, Université Pierre et Marie CURIE, may 1993.

[2] Anderson, T. E., Bershad, B.Ñ., Lazowska, E. D., and Levy, H. M. Scheduler activations: effective kernel support for the user-level management of parallelism. In *Proceedings of the13th ACM Symposium on Operating Systems Principle* (Oct. 1991), ACM SIGOPS, pp. 95–109. Published in ACM Operating Systems Review Vol.25, No.5, 1991. Also published ACM Transactions on Computer Systems Vol.10 No.1, pp.53–70, Feb 92.

[3] Arcia, A., and León, L. Técnicas de localización de objetos móviles. Tech. Rep. RT-CEM-100-2000, CEMISID - Universidad de Los Andes, 2000.

[4] Artsy, Y., and Finkel, R. Designing a process migration facility - the charlotte experience. *IEEE Computer 22*, 9 (1989), 47–56.

[5] Chase, J. S., Levy, H. M., Feeley, M. J., and Lazowska, E. D. Sharing and protection in a single-address-space operating system. *ACM Transactions on Computer Systems 12*, 4 (Nov. 1994), 271–307.

[6] Chasse, J., Amador Franz, G., Lazowska, E., Levy, H., and Litlefield, R. The amber system: parallel programming on a network of multiprocessors. In *12th ACM Symposium on Operating Systems Principles* (dec 1989), pp. 147–158.

[7] Douglis, F., and Ousterhout, J. K. Process migration in the sprite operating system. In *7th International Conference on Distributed Computer Systems* (sep 1987).

[8] Group, O. M. *The Common Object Broker: Arquitecture and Specification*, 2.3 ed., jul 1999.

[9] Jul, E., Levy, H., Hutchinson, N., and Black, A. Fine-grained mobility in the emerald system. *ACM Transaction on Computer Systems 6*, 1 (feb 1988), 109–113.

[10] León, L. Migracion non préemptive dans cool. In *Journées de Recherche du GDR-PRS-CNRS sur la Mémoire Partagée Répartie* (may 1996), pp. 2–14.

[11] León, L. *Une architecture pour les systèmes répartis à objets mobiles.* PhD thesis, Université Paris VI, feb 1998.

[12] Milojicic, D., Zint, W., Dangel, A., and Giese, P. Task migration on the top of the mach microkernel. In *3rd Usenix Mach Symposium*, pp. 273–290.

[13] Powel, M., and Miller, B. Process migration in DEMOS/MP. In *9th Symposium on Operating Systems Principles* (oct 1983), pp. 110–119.

[14] Richmond, M., and Hitchens, M. A new process migration algorithm. *Operating Systems Review 31*, 1 (1997), 31–42.

[15] Roush, E. T. *The Freeze Free Algorithm for Process Migration.* PhD thesis, Univsersity of Illinois at Urbana-Champaign, 1995.

[16] Rozier, M., Abrossimov, V., Armoand, F., Boule, I., Gien, M., Herrmann, F., Kaiser, C., Leonard, P., Langlois, S., and Neuhauser, W. The chorus distributed operating system. *Computing Systems 1* (oct 1988), 305–379.

[17] Saltzer, J. H., Reed, D. P., and D., C. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems 2* (nov 1984), 277–288.

[18] Shapiro, M., Gourhent, I., Habert, S., Mosseri, L., Ruffin, M., and Valot, C. Sos: An object-oriented operating system - assesment and perspectives. *Computing Systems 2*, 4 (1989), 287–337.

[19] Steketee, C., Zhu, W., and Moseley, P. Implementation of process migration in amoeba. In *14th International Conference on Distributed Systems* (1994), pp. 194–203.

[20] Theimer, M. M., Lantz, K. A., and Cheriton, D. R. Preemtable remote execution facilities for the v-system. In *10th ACM symposium on Operating Systems Principles* (dec 1985), pp. 2–14.

[21] Zayas, E. R. Attacking the process migration bottleneck. In *11th ACM symposium on Operating Systems Principles* (nov 1987), pp. 2–14.