Localización de objetos distribuidos móviles *

Andrés Arcia - Leandro León
Universidad de Los Andes
Centro de Estudios en Microelectrónica y Sistemas Distribuidos (CEMISID)
Mérida, Venezuela, 5101.
amoret@cemisid.ing.ula.ve
lrleon@cemisid.ing.ula.ve

Resumen

En un sistema distribuido a objetos, la localización de un objeto es el acto de actualizar referencias distribuidas cada vez que el objeto migra de un sitio a otro. Aunque muchas técnicas de localización han sido propuestas, su desempeño se degrada conforme aumenta la escala del sistema.

Este trabajo identifica y desarrolla un conjunto de técnicas para localizar objetos móviles en un sistema distribuido. Las técnicas en cuestión son: difusión, caching, prefetching y piggybacking. El uso cooperativo de estas técnicas permite escalar la localización a grandes cantidades de objetos, sitios y distancias de los sistemas distribuidos del futuro. La efectividad de las técnicas es evaluada a través de una implementación real y medidas de desempeño.

Palabras claves: sistemas distribuidos, objetos distribuidos, localización, identificación, migración, movilidad.

Abstract

In object oriented distributed systems, object location is refered as updating distributed references. This updating is probably done every time an object migrates. Even when several location techniques have been proposed, they show poor performance when scale grows.

Our proposal presents and develops a set of techniques for mobil object location in distributed systems. These techniques are: multicasting, caching, prefetching and piggybackcing. Their cooperative use allows to grow the scale in objects, sites and distance without any lose of performance. We have evaluated the efectiveness through an implementation and performance statistics.

Keywords: distributed systems, distributed objects, location, migration, mobility, identification

1. Introducción

Los objetos distribuidos se están convirtiendo en el paradigma de diseño y desarrollo de sistemas distribuidos. Una clara indicación de ello la constituye la popularidad de Java [6, 7] y CORBA [8] como modelos de desarrollo de sistemas distribuidos.

En el contexto de objetos distribuidos, la migración de un objeto es el acto de mover un objeto entre los sitios o procesos que conforman el sistema. Este mecanismo tiene varios atractivos, entre los que se encuentran el equilibrio de carga, la administración del sistema y la programación de mecanismos especiales asociados a abstracciones de distribución. La migración ha sido propuesta desde el inicio de los sistemas distribuidos a objetos, pero no es hasta tiempos recientes que se han desarrollado mecanismos efectivos.

En los sistemas a objetos, el acceso a un objeto, junto con sus privilegios, es realizado a través de una referencia al objeto. En un sistema distribuido, es natural y esperado que existan varias referencias repartidas a través de los sitios del sistema. Cuando un objeto migra, la referencias devienen inválidas y deben ser actualizadas. Esta actualización es lo que se conoce como la localización de objeto; es decir, localizar sus nuevas coordenadas físicas después de la migración.

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

1.1. Antecedentes de localización y sus problemas

Los enfoques primigenios para localizar objetos están basados en la difusión. En este sentido, dos técnicas han sido usadas. La primera consiste en difundir un mensaje global anunciando la nueva dirección cada vez que se realice una migración [12, 16, 1]. La segunda consiste en difundir un mensaje global de búsqueda cada vez que se detecte una referencia inválida [9]. Aunque la difusión es bastante eficaz para localizar objetos, sus costes son prohibitivos en sistemas de alta escala.

La técnica más popular es denominada “lazos de persecución” (forward adresses) [12, 9, 14, 4, 2, 15, 13, 3]. Cada vez que un objeto migra, se deja en el sitio origen una indicación -el lazo- acerca de la nueva localidad del objeto. Cuando llega un mensaje hacia la antigua localidad, éste es redireccionado hacia la localidad señalada por el lazo.

Los lazos de persecución han sido usados con éxito en sistemas de mediana escala. Lamentablemente, los lazos conllevan problemas serios cuando se aumenta la escala. En primer lugar, el acceso a un objeto se degrada conforme aumenta la cantidad de migraciones y, por ende, la longitud de la cadena de lazos. La cadena misma plantea un serio problema para la tolerancia a fallas del sistema, pues una falla en cualquiera de los sitios de una cadena compromete todas las referencias anteriores al lazo fallante. Además, un lazo consume recursos que deberían de ser liberados cuando se detecte que no hay más referencias apuntando al lazo. Esto conlleva a algoritmos de detección de lazos innecesarios y de limpieza de los mismos; en otras palabras, un mecanismo distribuido de “garbage collection”.

Para paliar los problemas asociados a los lazos de persecución, Shapiro et al [11, 5] han propuesto e implementado un protocolo de redundancia de lazos que evita la sensibilidad a fallas asociada a los lazos de persecución [11]. En añadidura, su mecanismo actualiza las referencias cuando se recorre una cadena de lazos, lo que solventa la degradación de desempeño. Posteriormente, este grupo desarrolló un mecanismo enteramente distribuido que detecta lazos no usados y los elimina del sistema [5]. Los trabajos de Shapiro et al constituyen un hito y un enorme aporte a la algorítmica distribuida y al desarrollo de sistemas distribuidos. A pesar de su valía intelectual, sus técnicas no evaden los costes en performance asociados a la eliminación distribuida de los lazos y no evitan que un acceso recorra una cadena larga de lazos con su consecuente degradación de desempeño.

1.2. Los objetivos de este trabajo

En este trabajo argüimos que es posible lograr una localización de alto desempeño para sistemas de gran escala sin utilizar lazos de persecución. Mediante la utilización de diversas técnicas, hemos llevado a cabo una implantación completa de un sistema de localización que satisfizo los siguientes requerimientos iniciales de diseño:

- **Alto desempeño**: independientemente de la escala, el sistema debe actualizar referencias rápidamente, con la menor cantidad de mensajes de red.
- **Independencia de la escala**: aumentos significativos de la escala, no deben impactar el alto desempeño de la localización.
- **Portabilidad**: el sistema debe ser fácilmente portátil a lo largo de las diversas plataformas materiales y sistemas operativos.
- **Tolerancia a fallas**: la localización debe soportar caídas de sitios que no estén directamente involucrados en el acceso a un objeto.

1.3. La propuesta

Los objetivos que acabamos de mencionar son fácilmente alcanzables si se sigue una estrategia de localización perezosa que denominamos “actualización por fracaso de acceso” o, en corto, “actualización por fracaso”. Fundamentalmente, esta estrategia se resume en los siguientes lineamientos:

1. Existe un servicio de localización encargado de localizar objetos. En concreto, este servicio debe satisfacer dos servicios abstractos. El primero es el registro de objeto: cuando un objeto se crea o se mueve, su dirección es registrada en el servicio de localización. El segundo servicio es la búsqueda, la cual consiste en retornar la dirección del objeto.

   Desde la perspectiva de la interfaz, el servicio de localización es centralizado. Desde la perspectiva de implementación, el servicio es distribuido.

2. Cuando desde una referencia inválida se accede a un objeto, el acceso causa una excepción. Este evento provoca una solicitud de búsqueda al servicio de localización, el cual responde con la nueva dirección del objeto. La referencia es actualizada y la invocación es repetida. La detección de la excepción es embebida en el código stub de la referencia, lo que permite una localización completamente transparente.

A nivel de interfaz, este enfoque es muy práctico, pues no requiere sincronización ni garbage collection. Sobre un servicio de localización centralizado, este enfoque sólo gasta tres mensajes de red para acceder a un objeto recientemente migrado. Empero, un enfoque centralizado es muy sensible a fallas, no es escalable y es prohibitivo para largas distancias.

Para conjugar los beneficios a nivel de interfaz de la actualización por fracaso con un servicio escalable, de alto desempeño y tolerante a fallas, el diseño del servicio de localización es totalmente distribuido con un mantenimiento intensivo caches de referencias recientes.

Nuestro servicio estará basado en servidores de localización. Cada servidor mantiene una "tabla propietaria" que almacena localidades de objetos. Cuando un objeto migra, sus coordenadas pueden ser eliminadas de la tabla propietaria de un servidor y guardadas en la tabla propietaria de otro. En todo caso, la tabla propietaria ofrece información correcta de la localización de un objeto.

Resultados de búsquedas previas se guardan en los caches. De este modo, requerimientos de búsqueda pueden ser satisfechos sin necesidad de encontrar el servidor propietario del objeto. Para que el uso de caches sea efectivo, es necesario:

- Que las localidades de los objetos redunden a través de los caches de los diversos servidores.
- Que la información contenida en el cache sea vigente; es decir, que no corresponda a búsqueda de objetos que migraron de nuevo.

La información contenida en los caches es redundante a través de los servidores del sistema. Esta redundancia ofrece un soporte para el procesamiento de fallas.

Requerimos, entonces, de políticas que refresquen los caches con información reciente y pertinente. Además, necesitamos ser capaces de recuperar la dirección de un objeto si, por mala suerte, los caches y la tabla propietaria fallan. A tales efectos, contamos con las siguientes técnicas:

1. **Prefetching**: cuando ocurre una migración de objeto, algunos servidores pueden ser notificados de la nueva localidad. De este modo, los caches son refrescados con la información más reciente.

2. **Piggybacking en invocaciones**: la invocación es una actividad muy frecuente y fundamental en un sistema a objetos. Podemos, entonces, aprovechar los mensajes de invocación para intercambiar información entre los servidores de localización.

   Una indicación de localidad implica información de corto tamaño. Así pues, el impacto de este intercambio no debería ser muy grande. En añadidura, los protocolos de red y de transporte actuales trabajan en función de un grano de mensaje. La latencia y ancho de banda de tales protocolos no es exactamente lineal. Para tamaños de mensajes aproximados, pero no iguales, el desempeño del protocolo es el mismo.

3. **Protocolo de recuperación**: Si a pesar de las técnicas anteriores no es posible recuperar una localidad, entonces es necesario encontrar el servidor propietario. Para ello, como último recurso, establecemos un protocolo que efectúa difusiones en cascada, desde la de menor hasta la de mayor costo. En líneas generales, tal protocolo consiste en:

   a) Se envía un mensaje no fiable preguntando por el localizador propietario del objeto, quien debería manifestarse con un mensaje fiable de vuelta.

   b) Si el paso anterior fracasa, se difunde un mensaje no fiable a todos los servidores. Cualquier servidor que encuentre información sobre el objeto responde con un mensaje fiable.

   c) Si el paso anterior fracasa, se envía un mensaje fiable preguntando por el localizador propietario del objeto, quien debería manifestarse con un mensaje fiable de vuelta.

   d) Si el paso anterior fracasa, entonces se difunde un mensaje fiable ordenando a todos los servidores reconstruir su tabla propietaria. Consecuentemente, si el objeto existe, el localizador propietario responde un mensaje fiable con la localidad.

   e) Si el paso anterior fracasa, el objeto es declarado inexistente.

2. Localización de alto desempeño

Cuando un objeto o proceso migra, todas sus referencias quedan inválidas. Entonces, cualquier acceso desde alguna de éstas referencias fracasará. En todo caso, el fracaso se hará saber a través de un mensaje de respuesta.

Existen dos escenarios posibles cuando se efectúa una invocación. Primero, la invocación puede tener éxito, el método remoto es ejecutado y una respuesta es recibida. Segundo, la invocación puede fracasar. El fracaso será entonces indicado mediante cualquiera de las siguientes respuestas:

1. *"El objeto no fue encontrado"*: Con esta respuesta se concluye que el sitio destino de la invocación no conoce el objeto. Existe todavía la posibilidad de que se encuentre en cualquier otro sitio del sistema.
2. *"Existe una referencia más reciente"*: En este caso, el sitio destino sabe que el objeto no está presente, pero conoce una referencia, proporcionada por el cache, más reciente hacia el objeto. Esto sugiere la posibilidad de reinvocar desde el sitio origen con esta nueva dirección.
3. *"El objeto está eliminado"*: Cuando se obtiene una respuesta de este tipo no será posible ejecutar la invocación, pues el objeto ha sido eliminado del sistema.

En adelante, estos mensajes serán referidos como mensajes de fracaso de tipo 1, tipo 2 y tipo 3.

Al ocurrir un fracaso, se solicita al servicio de localización la nueva localidad del objeto. El servicio de localización se encargará, entonces, de encontrar al objeto y de notificar su dirección al ente solicitante. Una vez encontrado el objeto, la invocación es repetida.

La arquitectura del servicio de localización es enteramente distribuida. En cada sitio existe un proceso especial de localización, en adelante "localizador", que registra todos los objetos y procesos del sitio y, a través de caches mantiene información reciente acerca del paradero de los objetos más recientemente utilizados. También existe una parte ejecutiva que va añadida a cada proceso cliente del localizador y es denominado "localizador proceso", cuya función es la de gestionar la comunicación con el servicio de localización.

Un punto crucial en este estudio es la capacidad que tiene un localizador para revisar e intervenir accesos entrantes o salientes. Cuando una referencia efectúa un acceso, el mensaje saliente de red es inspeccionado por el localizador del sitio origen, el cual puede modificar o redireccionar el mensaje. Del mismo modo, en el sitio destino del acceso, el localizador inspecciona el mensaje, el cual también puede modificar.

2.1. Caching

El caching consiste en almacenar indicaciones referentes a la ubicación de los objetos. Una indicación se obtiene a través de búsquedas previas o de mensajes de actualización. Eventualmente, la información en los caches podrá utilizarse en futuras búsquedas.

Los caches representan una manera de amortizar los costes inherentes a las difusiones. Diversos tipos de caches estarían habilitados en cada localizador para almacenar cálculos pertinentes a una localización.

Una cache tiene una capacidad finita. Cuando un cache está lleno y se requiere insertar nuevas entradas, el espacio para estas nuevas entradas es asignado bajo una política del menos recientemente utilizado. La entrada en cada uno de los caches es débil en el sentido de que puede ser incorrecta. Esto sucede, por ejemplo, cuando un objeto migra dos veces y el cache mantiene el registro de la primera migración.

2.1.1. Cache de migración

En el cache de migración se guardan las nuevas direcciones que los objetos dejan al migrar. Este cache es una especie de lazo de persecución temporal. Dado un objeto O que migra del sitio x al sitio y, su nueva dirección en y es anotada en el cache de migración del sitio x. Así, futuras invocaciones hacia O que lleguen al sitio x serán respondidas con un mensaje de fracaso del tipo 2, obtenido mediante la información del cache.

Un registro en este cache contiene tuplas de forma $< A, z, t >$. A es un identificador de objeto y z es el último sitio válido conocido donde estuvo A en el tiempo lógico t. En adelante t es una estampilla de tiempo lógico que indica el número de migraciones del objeto. Esta estampilla impone un orden en las direcciones según su tiempo de migración que permite comparar un par de referencias y descartar la más antigua.

2.1.2. Cache de entrada

El cache de entrada guarda referencias remotas correspondientes a accesos entrantes exitosos.

Un registro de este cache contiene la 3-tupla $< O, x, t >$. O es un identificador de objeto invocado, x es el identificador de sitio donde existe una referencia hacia O, y t es el tiempo de la última modificación de la entrada en el cache.

2.1.3. Cache de salida

Dado un localizador, el cache de salida almacena referencias correspondientes a accesos salientes. Toda referencia utilizada para invocar es contrastada por el localizador con la información almacenada en este cache. Si el localizador encuentra una referencia más reciente en el cache de salida, entonces es sustituida.

El cache de salida es el que está más propenso a actualizaciones, pues, por definición, es donde se puede encontrar con mayor probabilidad una referencia válida.

Un registro de éste cache contiene la 3-tupla $< A, x, t >$, que corresponde a una referencia. A es un identificador de objeto y x es un sitio donde residió el objeto en el tiempo lógico t.

2.1.4. Cache de nuevas referencias

Mediante piggybacking, un localizador puede llegar a conocer de forma adelantada la nueva dirección de un objeto. Supongamos que un localizador l_x en el sitio x recibe un mensaje notificándole: "*Se ha determinado una nueva referencia L para el objeto O*". El localizador busca en el cache de salida una entrada para el objeto O. Si la referencia L es más reciente, entonces el cache de salida es actualizado con L. No obstante, si no se encuentra una referencia en el cache de salida, L es anotada en el cache de nuevas referencias. Este cache mantiene referencias que podrían ser útiles en futuras invocaciones.

Un registro de éste cache mantiene referencias formadas por la 3-tupla $< A, x, t >$. Donde, A es un identificador de objeto y x es un sitio donde residió el objeto en el tiempo lógico t.

2.1.5. Cache de eliminaciones

Cuando un objeto es eliminado del sistema, su identificador único queda registrado en este cache. Las entradas en este cache, pueden ser propagadas al resto de los caches.

2.2. Prefetching

En este trabajo, prefetching es la acción de actualizar una referencia inválida antes de que ocurra un acceso. El sentido de esta técnica es evitar el costo del fracaso de acceso.

Para determinar los sitios hacia donde se puede realizar el prefetching, se utilizan los caches de los sitios origen y destino de migración.

2.2.1. Prefetching desde el sitio origen de la migración

El sitio origen de la migración puede activar el prefetching de las siguientes maneras:

Con el cache de entrada : se buscan referencias hacia el objeto migrante y se les notifica a los sitios remotos la nueva localidad del objeto.

Con el cache de migración : cuando en el antiguo sitio arribe una invocación hacia el objeto migrado, el localizador encontrará en el cache de migración una referencia más reciente. Entonces, se responderá al sitio invocante con un mensaje de fracaso del tipo 2 que lleva una referencia más reciente.

2.2.2. Prefetching desde el sitio destino de la migración

Para que el sitio destino de la migración pueda efectuar una futura acción de prefetching, deben efectuarse las siguientes operaciones con los caches:

Con el cache de entrada : Las referencias en el cache de entrada referentes al objeto migrante pueden copiarse hacia el cache de entrada del sitio destino. De esta manera, el sitio destino puede efectuar prefetching tal como se describió en § 2.2.1.

Con el cache de salida : Al arribo del objeto al sitio destino, se busca en el cache de salida una referencia hacia el objeto migrante. Si ésta es encontrada en el cache, entonces se actualiza: la nueva referencia pasa a ser local.

2.2.3. Prefetching explícito con el cache de salida

Las entradas del cache de salida son periódicamente verificadas. La técnica consiste en enviar un mensaje cada t unidades de tiempo, preguntando por la validez de las referencias. Si alguna referencia es inválida, entonces se inician acciones de búsqueda del objeto con la esperanza de encontrarlo antes de que ocurra una invocación.

2.3. Piggybacking

Un mensaje "piggyback" consiste en incluir un mensaje pequeño al cuerpo de otro mensaje. De este modo, los localizadores intercambian información que les permite actualizar sus caches en background.

Los mensajes piggybacks son usados bajo las suposiciones siguientes:

- Todo mensaje de red, inclusive los de requerimiento y de respuesta de las invocaciones, es utilizado por el localizador para enviar piggybacks.
- El localizador puede propagar los piggybacks. Si un localizador recibe un piggyback, éste puede aprovechar mensajes de red y enviarlo a otros localizadores.
- Para todo piggyback se conoce el sitio desde donde se inició la búsqueda.
- Un piggyback puede ser identificado unívocamente por el identificador del objeto al que éste se refiere.
- Los piggybacks referentes a un objeto, serán actualizados conforme llegue información más reciente o de mayor prioridad.

2.3.1. Tipos de piggyback

Los siguientes piggybacks tienen como principal función actualizar referencias:

1. **Búsqueda de objeto**: indica la necesidad de búsqueda o actualización de una referencia. Este mensaje esta compuesto de una referencia que requiere ser actualizada.

   Cuando un localizador l_x recibe una solicitud de búsqueda del objeto O, l_x intercepta invocaciones salientes del sitio y les incluye implícitamente un mensaje piggyback que llamaremos "*se busca*".

   Cuando un localizador cualquiera l_z recibe éste mensaje, l_z busca el objeto O en su tabla propietaria. Si encuentra una dirección más reciente, entonces l_z envía un mensaje explicito a l_x notificándole la nueva dirección del objeto. Esta contestación podría ser a través de otro piggyback si está partiendo un mensaje explícito de invocación hacia x.

   Si l_z no encuentra el objeto O en su tabla propietaria de objetos, entonces l_z busca en sus caches una dirección más reciente. Si lo encuentra, entonces l_z envía un mensaje a l_x con la nueva referencia. Además, sigue propagando el mensaje.

2. **Anuncio de Referencia**: propaga un evento de actualización de una dirección. Una dirección puede ser considerada nueva en dos situaciones: en el nacimiento de un objeto o, cuando un objeto migra.

   Cuando un objeto ingresa por primera vez al sistema, existe la posibilidad de que varias referencias requieran información sobre el sitio de residencia del nuevo objeto.

   Por otro lado, cuando un objeto migra, su estampilla de tiempo lógico es incrementada en una unidad. Esto, conjuntamente con el nuevo sitio, hacen cambiar la referencia. Las referencias antiguas muy probablemente requieran de esta nueva información. También se utiliza cuando el objeto referenciado no es conocido por el sitio y se almacena en el cache de nuevas referencias.

3. **Eliminación de Objeto:** Cuando un objeto A es eliminado del sistema, su eliminación es anunciada mediante el piggyback: "*El objeto A fue eliminado*". A este mensaje lo llamaremos "*Eliminado*".

   Cuando un localizador recibe un "*eliminado*", el localizador elimina todas las entradas de sus caches que contengan al objeto y coloca el mensaje "*eliminado*" en el cache de eliminación. Luego el localizador puede propagar el mensaje a otros localizadores.

2.4. Control de congestión de piggybacks

Puesto que los piggybacks consumen recursos de comunicación, es necesario emplear mecanismos que determinen cuando cesar la transmisión de un piggyback.

Una primera manera de anular la propagación de un piggyback es la recepción de otro piggyback que involucre al mismo objeto pero que represente un evento más reciente o de mayor prioridad.

En principio, un localizador mantiene información acerca de los sitios que ya conocen o conocerán un piggyback p_i en particular. Dado un sitio x, el localizador l_x puede asumir que un piggyback p_i es conocido por el localizador l_y en el sitio y, si p_i ha sido enviado desde x hacia y. Por otra parte, si el localizador l_y recibe el mensaje p_i desde el sitio x, se asume que x ya conoce y tiene en sus tablas a p_i.

2.4.1. Grafo de frecuencia de invocaciones

Los caches de entrada y salida de los localizadores pueden ser usados para construir un mapa actual de invocaciones en la red. Este mapa estaría construido en base a las invocaciones entrantes y salientes más recientes. Estas se encuentran almacenadas en una estructura de datos especial que en parte se comporta como un cache, guardando la información de cuales son los mensajes y los sitios donde son conocidos.

Según el mapa, cada localizador mantiene una lista "esperanza" de los sitios del cual va a recibir piggybacks. Tomando en cuenta un criterio estadístico, un sitio puede decidir no enviar piggybacks hacia aquellos sitios para los cuales se espera recibir invocaciones.

Otra lista "esperanza" mantiene los sitios al cual el localizador enviará piggybacks. Este método requiere anotar el punto de partida de los piggyback. Este grafo ayuda al localizador a decidir a cuáles sitios propagar los piggyback recibidos.

2.4.2. Contador máximo de uso de piggyback

Cada piggyback tiene un contador asociado. Este contador es incrementado cada vez que un mensaje llega a un sitio diferente o, cuando es considerado para ser enviado pero por falta de espacio en el buffer de piggybacks no es enviado. Así, cada localizador maneja un máximo que anula la propagación.

2.4.3. Tiempo de vida los piggybacks

Los piggybacks tiene un tiempo de vida físico en el sistema. Este tiempo de vida se fija cuando el mensaje es creado y posteriormente, si es requerido, puede ser actualizado durante su propagación. Este lapso consiste en la diferencia entre la fecha de creación del mensaje y la fecha local en el sitio de llegada al arribo del piggyback.

2.4.4. Máxima cantidad de piggybacks por mensaje de red

Los piggybacks que viajan en un mensaje de red son limitados por dos factores: el primero es una cantidad máxima de piggybacks por mensaje de red, impuesta según criterios de performance. El segundo, es que el tamaño de un mensaje no debe pasar el tamaño del MTU (Maximum Transfer Unit); entonces, un mensaje cualquiera puede completarse con piggybakcs hasta no sobrepasar el tamaño del MTU.

2.5. Difusiones

A pesar de que los caches, prefetching y piggybacks han comprobado su eficacia para la localización de objetos, estas técnicas, al igual que cualquier otra, pueden fallar. En situaciones de fracaso de las técnicas anteriores, se debe efectuar una búsqueda exhaustiva mediante difusiones.

En su expresión más simple, el sitio que desea localizar un objeto efectúa una difusión y espera por la respuesta del localizador propietario del objeto.

Para disminuir y amortizar los costes se consideran dos protocolos: difusión débil y difusión fuerte. En la difusión débil la entrega de los mensajes no es garantizada. En contraposición, la difusión fuerte garantiza la entrega de los mensajes de solicitud de búsqueda a cada sitio. La difusión fuerte tiene asociado un costo mayor que la difusión débil.

2.6. Protocolo de difusiones débiles

1. Efectuar una difusión débil solicitando respuesta del localizador propietario.
2. Si la difusión anterior no logra encontrar el objeto, entonces efectuar una difusión débil ordenando a cada localizador recabar toda la información que cualquier localizador tenga del objeto solicitado. La información es devuelta con mensajes fiables.

2.7. Protocolo de difusiones fuertes:

1. Efectuar una difusión fuerte exigiendo respuesta del localizador propietario.
2. Si el paso anterior fracasa, entonces se difunde un mensaje fiable ordenando a todos los servidores reconstruir su tabla propietaria.
3. Si el paso precedente no logra localizar el objeto, entonces el objeto es declarado "inexistente".

Este protocolo es utilizado como ultimo recurso; es decir, cuando otros métodos menos costosos no logren localizar al objeto.

3. Implantación

En la figura 1, se aprecia la arquitectura general del sistema localizador. De manera general esta conformado por un módulo de comunicación local (entrada-salida local), un módulo de comunicación remota (entrada-salida remota) y las estructuras de datos. El módulo de comunicación local procesa llamadas al localizador, y el módulo de comunicación remota, procesa mensajes explícitos recibidos.

La ejecución de estos módulos tiene una interacción directa con las estructuras de datos donde quedan registrados los eventos del localizador: tablas, caches y contenedor de piggybacks.

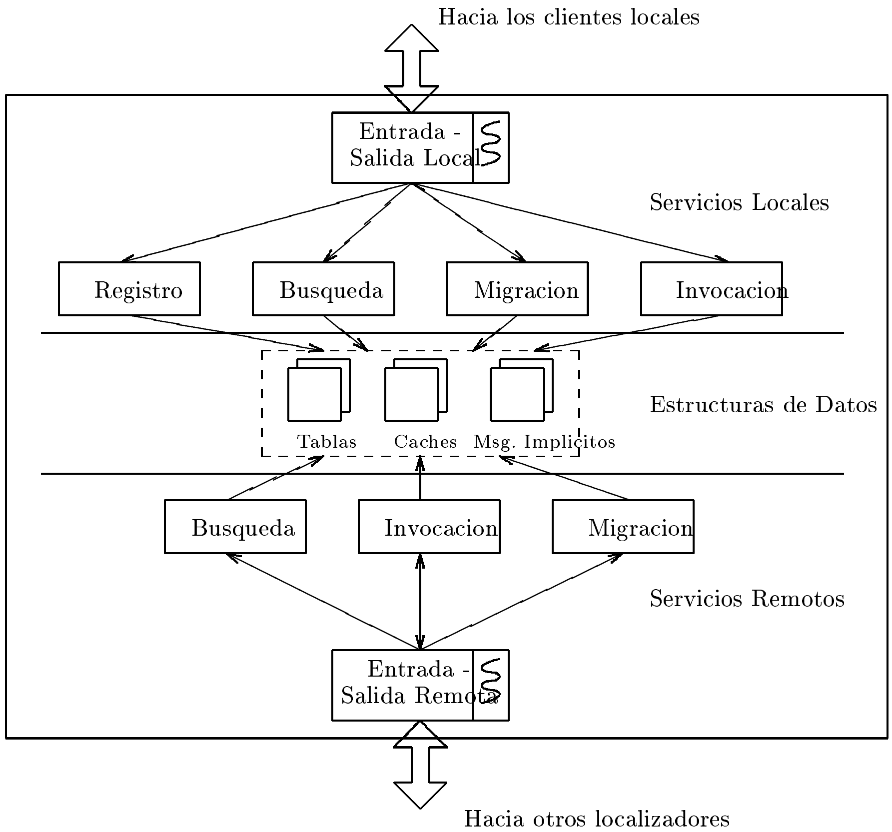

Figura 1: Arquitectura del localizador

3.1. Módulo de entrada y salida local

Este módulo resuelve el problema de comunicación local fiable entre los procesos y el sistema localizador. Este módulo procesa los servicios de registro, migración, invocación y localización.

3.2. Módulo de entrada y salida remota

Este módulo es responsable de la comunicación entre localizadores. Utiliza el sistema IPC [10] de $\mathcal{ALEPH}$, que ofrece una semántica de *exactamente una vez* para la entrega de mensajes al puerto destino. Ofrece también funciones que optimizan la entrega de invocaciones y la entrega de mensajes simples.

3.3. Mensajes del localizador

Un mensaje de localizador tiene como función anunciar un evento de un localizador a otro con la finalidad de realizar una invocación o de ejecutar una acción de localización. Estos mensajes son generados como consecuencia de la petición de alguno de los servicios que el sistema localizador exporta a los clientes. En cada uno de estos mensajes, existe una sección destinada a llevar piggybacks. El tamaño de esta sección es determinado en tiempo de ejecución y según criterios de performance.

Se tiene un total de 8 mensajes de localizador que se describen a continuación:

1. **Solicitud de invocación**: mediante este mensaje se hace la invocación a un objeto. Este mensaje tiene información suficiente para identificar el origen y destino de la invocación.

2. **Respuesta a la invocación**: este mensaje sirve para transportar la respuesta a una invocación.

   Existen varias maneras de responder a un mensaje de invocación: invocación exitosa, objeto no encontrado, binding más reciente y objeto eliminado.

3. **Solicitud de búsqueda**: es el mensaje enviado en una difusión.

4. **Anuncio de inexistencia de un objeto:** Dice si un objeto ha sido eliminado del sistema. Lleva el identificador del objeto eliminado.

5. **Indagación de existencia**: permite saber si un objeto todavía esta en el lugar donde dice su *"referencia"*. Es el mensaje utilizado para hacer el prefetching con el cache de salida.

6. **Respuesta a la indagación**: ofrece información acerca del estado del objeto indagado con el mensaje anterior. Es el mismo tipo de respuesta que fue descrito en el mensaje **Respuesta a la invocación**.

7. **Anuncio de referencia**: Este mensaje tiene como función llevar una referencia hacia cualquier sitio. El mensaje es utilizado en acciones de prefetching y difusión.

8. **Registros del cache de entrada**: lleva al localizador destino de la migración las entradas correspondientes al objeto en el cache de entrada del sitio origen de la migración. Esta formado por una lista de identificadores de sitio que se refiere a las direcciones de los localizadores desde donde se hicieron invocaciones al objeto migrante.

El sistema está enteramente implementado. sus fuentes pueden ser descargados y examinados para verificación en:

`ftp.cemisid.ing.ula.ve/pub/aleph/locator`

4. Desempeño

Se realizó un experimento con 5 sitios, 2 procesos por sitio, 5 objetos por proceso y 5 referencias por proceso. Esto totaliza 50 objetos y 50 referencias en todo el sistema, la diferencia de escala está dada por una cuantificación de un orden de magnitud de diferencia entre el número de sitios y la cantidad de objetos. Los objetos fueron programados para moverse por todo el espacio de sitios según una probabilidad. Las referencias fueron escogidas aleatoriamente y siempre hacia objetos remotos.

La invocación ocurría según una probabilidad p y la migración según una probabilidad q, donde $p+q=1$. Se trabajaron dos valores de p y q para tres distintos experimentos.

En el Cuadro 4 se presenta el resultado de los experimentos. Los porcentajes fueron calculados en base a un total de 5000 eventos distribuidos según p y q.

Observando los experimentos 1 y 2 se nota que a medida que aumenta la cantidad de migraciones existe una mayor tasa de reinvocación; pues es más probable que la información en los caches sea incorrecta.

Por otro lado, la comparación del experimento 2 y 3 denota la incidencia que promete el piggyback en la localización de objetos móviles. Pues, se observa como la proporción de reinvocaciones disminuyo en aproximadamente un 50 %.

Es importante recalcar que en ningún experimento hizo falta utilizar difusiones para conseguir un objeto.

experimento	*técnicas usadas*	p	q	*reinvocaciones*	*exitos*
1	caching y prefetching	90 %	10 %	0,8 %	99,2 %
2	caching y prefetching	70 %	30 %	5,62 %	94,38 %
3	caching, prefetching y piggybacks	70 %	30 %	2,7 %	96,3 %

Cuadro 1: Performance del Sistema de Localización.

5. Conclusiones

La localización de objetos tal y como fue tratada en este trabajo, tiene una connotación completamente distribuida. Esto hace que buscar un objeto sea una tarea cooperativa, donde se puede prescindir de algunos de los localizadores sin que esto afecte la calidad del servicio gracias a la redundancia de información ofrecida por los caches.

El servicio de localización también muestra versatilidad, pues está ampliamente parametrizado. Cambiando unas pocas líneas de código, un localizador puede servir a clientes remotos, conformando así un esquema con un poco de centralización si así lo quisiera el cliente.

El sistema es escalable por varias razones. Primero, la distribución planteada en el sistema permite escalarlo, pues a mayor numero de sitios, mayor será la probabilidad de encontrar información redundante sobre el paradero de los objetos. Segundo, el sobrecoste del envío y recepción de invocaciones a través del localizador sobre redes distintas es despreciable.

El sistema garantiza la transparencia a través de una interfaz que ofrece la ilusión de centralización. El localizador es capaz de intervenir las invocaciones y redirigirlas, si en alguno de sus caches hay información más reciente del objeto invocado. Esto último hace al localizador transparente respecto al cliente.

Referencias

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

[2] Jeffrey Chasse, G. Amador Franz, Edward Lazowska, Henry Levy, and Richard Litlefield. The amber system: parallel programming on a network of multiprocessors. In *12th ACM Symposium on Operating Systems Principles*, pages 147–158, dec 1989.

[3] Mossière Jacques Chevalier Pierre-Yves, Hagimot Daniel and Xavier Rousset de Pina. Le système réparti à objets Guide. Technical report, 1995.

[4] Frederik Douglis and Jhon K. Ousterhout. Process migration in the sprite operating system. In *7th International Conference on Distributed Computer Systems*, sep 1987.

[5] Paulo Ferreira. *Larchant: ramasse-miettes dans une mémoire partagé répartie avec persistance par atteignabilité*. PhD thesis, Université Paris VI, may 1996.

[6] J. Gosling. Java intermediate bytecodes. 30(3):111–118, March 1995.

[7] James Gosling and Arthur van Hoff. Winner: Java. pages ??–??, December 1995.

[8] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, revision 2.2 edition, February 1998. (formal/98-07-01).

[9] Eric Jul, Henry Levy, Norman Hutchinson, and Andrew Black. Fine-grained mobility in the emerald system. *ACM Transaction on Computer Systems*, 6(1):109–113, feb 1988.

[10] Carlos Nava and León Leandro. Un sistema IPC para eldesarrollo de aplicaciones cliente-servidor en Internet. In *XXVI Conferencia Latinoamericana de Informática*, sept 2000.

[11] David Plainfossé. *Distributed Garbage Collection and Referencing Managment in the Soul System.* PhD thesis, Université Paris VI, jun 1994.

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

[13] Alexander Schill and Markus Mock. Dc++: Distributed object-oriented system support on top of OSF DCE. *Distributed System Engineering*, 1(2):112–125, dec 1993.

[14] Marc Shapiro, Ivon Gourhent, Sabine Habert, Laurence Mosseri, Michael Ruffin, and Céline Valot. Sos: An object-oriented operating system - assesment and perspectives. *Computing Systems*, 2(4):287–337, 1989.

[15] P. Sinha, K. Park, X. Jia, K. Shimuze, and M. Maekawa. Process migration in the galaxy distributed system. In *5th International Parallel Processing Symposium*, pages 611–618, 1991.

[16] Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemtable remote execution facilities for the v-system. In *10th ACM symposium on Operating Systems Principles*, pages 2–14, dec 1985.