Memoria técnica del proyecto

Programación de Sistemas Concurrentes y Distribuidos

2º curso, Grado de Ingeniería en Informática, 2020-2021

Número de equipo: *16*

Integrantes:

- *Javier Pardos Blesa, 698910*
- *Luis Palazón Simón, 795062*
- *Héctor Toral Pallás, 798095*
- *Carlos Paesa Lía, 798974*
- *Pablo Jesús Bueno Ereza, 799101*
- *Santiago Illa Criado, 805798*

Índice de contenidos

1. Introducción

1.1. Descripción del problema a resolver

El proyecto desarrollado se basa en un sistema de coordinación Linda distribuido, que ofrece las operaciones básicas de postNote (PN), removeNote (RN), readNote (RDN) y además, una operación de lectura doble, readNote múltiple (RDN_2), y una operación de lectura y borrado doble, removeNote múltiple (RN_2). El espacio de tuplas consta de tuplas planas de máximo 6 elementos, todos de tipo string. Los patrones usados por las operaciones de lectura pueden contener variables para ligarse a valores.

El sistema se divide en tres servidores (para trabajar con tuplas de tamaño 1-3, 4-5 y 6 respectivamente), a los que varios clientes pueden solicitar servicio mediante la librería Linda Driver. Para obtener la dirección IP y el puerto de los servidores Linda, los clientes deben consultar a un servidor registro de despliegue.

Adicionalmente, un servidor de monitorización muestra periódicamente información estadística del espacio de tuplas y además, todos los procesos involucrados en el sistema finalizarán controladamente su ejecución.

Este sistema Linda se aplica para resolver un problema de gestión de ofertas de viaje, en el que existen 5 procesos publicadores de ofertas (número de oferta, origen, destino y precio), 10 buscadores simples (de una oferta) y 5 buscadores de ofertas combinadas (pasando por una ciudad intermedia), con un máximo de 8 procesos buscando oferta en cada momento.

1.2. Participantes en el proyecto

El trabajo ha sido desarrollado por el grupo 16, compuesto por Javier Pardos Blesa (698910), Luis Palazón Simón (795062), Héctor Toral Pallás (798095), Carlos Paesa Lía (798974), Pablo Jesús Bueno Ereza (799101) y Santiago Illa Criado (805798), estudiantes de segundo curso de ingeniería informática de la Universidad de Zaragoza.

1.3. Organización de la memoria técnica

A continuación, se va a tratar en profundidad el diseño de la solución propuesta al problema, incidiendo en las decisiones más relevantes sobre la misma. Posteriormente, la planificación del proyecto y por último, su evaluación final, así como una breve conclusión.

2. Diseño de alto nivel de la solución

2.1. Figura explicativa

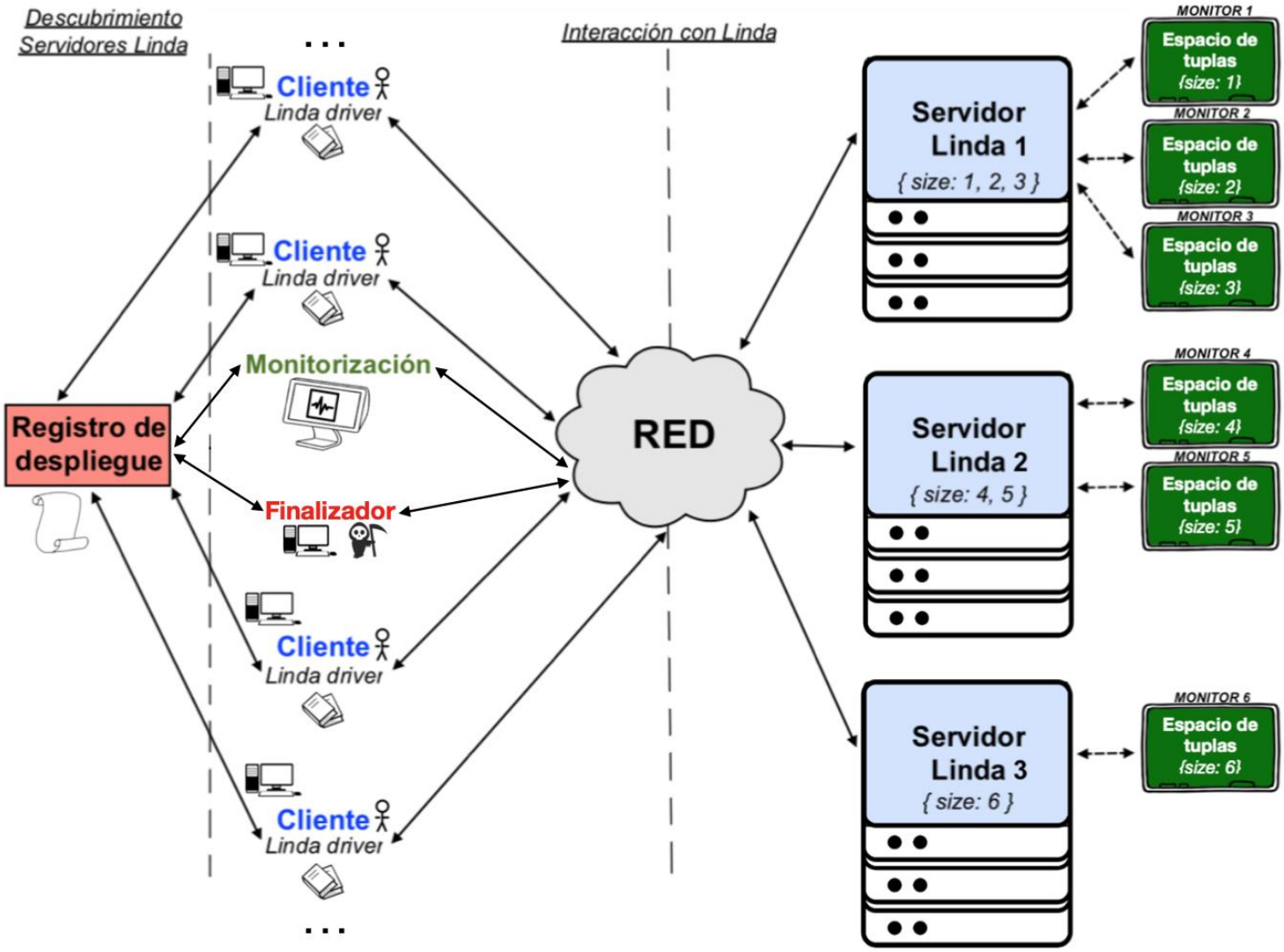


Figura 1: Esquema del sistema

2.2. Descripción del sistema

En la figura anterior se puede apreciar la estructura del sistema desarrollado. Dado que Linda es un sistema concurrente-distribuido, un cliente que quiera acceder a los servicios de Linda cuenta con la librería Linda Driver para poder interactuar con ellos de forma distribuida e independiente. El Linda Driver primero debe conocer la dirección IP y el puerto de los mismos. Para ello, se conecta a un servidor multithread llamado registro de despliegue, que le proporciona toda esta información sobre cada uno de los tres servidores Linda multithread. Así, el Linda Driver ya puede establecer la conexión con ellos. El cliente permanecerá conectado a los tres servidores durante toda su ejecución. Por cada operación que desee hacer el cliente, el Linda Driver envía dicha petición vía red al servidor correspondiente, en función del tamaño de las tuplas con las que desee trabajar (servidor 1: tamaño 1-3, servidor 2: tamaño 4-5, servidor 3: tamaño 6).

Por su parte, el servidor está bloqueado a la espera de recibir una petición de servicio. Cuando la recibe, el servidor identifica qué acción desea realizar el cliente y la efectúa en el espacio de tuplas correspondiente. Cada servidor, cuenta con un monitor de sincronización por cada tamaño de tupla con los que trabaja. Así, se garantiza la exclusión mutua en cada acceso individual a un espacio de tuplas de determinado tamaño. Tras esto, el servidor vuelve a bloquearse a la espera de otra petición de servicio.

Existe un servidor ajeno a Linda que se encarga de la monitorización, es decir, cada 20 segundos muestra por la salida estándar información estadística de cada espacio de tuplas. Para ello, al igual que un cliente normal, necesita conectarse al registro de despliegue para obtener la dirección IP y el puerto de los servidores Linda. Posteriormente, se conecta vía red a cada servidor y obtiene información de los espacios de tuplas asociados a cada uno. Esta información incluye el número de tuplas, lecturas pendientes, lecturas realizadas y escrituras realizadas en cada espacio.

Por último, hay un cliente especial, llamado cliente finalizador, cuya función es impedir a nuevos clientes conectarse al registro de despliegue y a los servidores Linda. Cuando se desee finalizar controladamente la ejecución de todos los procesos involucrados en el sistema, este cliente se conectará al registro de despliegue, recibirá las IP’s y puertos de los servidores Linda y le enviará un mensaje de finalización. Así, ningún cliente más podrá conectarse al registro de despliegue. Tras esto, el finalizador se conectará a cada servidor Linda y les enviará el mensaje de finalización, lo que impedirá al resto de clientes seguir accediendo a estos.

2.3. Descripción del sistema de gestión de ofertas de viaje

En el sistema de gestión de ofertas de viaje se pone en uso el sistema Linda realizado. El sistema Linda intercambiará, en base a sus protocolos, tuplas de tipo:

- id: Tupla de tamaño 1 que representa la ID de la oferta publicada.
- permiso: Tupla de tamaño 2 que representa un permiso de acceso al espacio de tuplas a un buscador. Todos los campos tienen el mismo elemento.
- fin de servicio: Tupla de tamaño 3 que representa el número de procesos buscadores que han finalizado en cada momento. Todos los campos tienen el mismo elemento.
- oferta: Tupla de tamaño 4 que representa una oferta de viaje. Tiene el siguiente formato: [ID_Oferta, Ciudad_Origen, Ciudad_Destino, Precio].

En la resolución del problema intervienen un gestor de publicadores y un gestor de buscadores. El gestor de publicadores se conecta al registro de despliegue, usando el Linda Driver, para obtener las IP's y puertos de los servidores Linda y así conectarse a ellos. Tras esto, actúa como publicador "init", es decir, publica en el espacio de tuplas 8 tuplas "permiso", una tupla "id" con valor inicial 1 y una tupla "fin de servicio" con valor inicial 0. Por último, lanza 5 threads publicadores que se conectan individualmente y de forma análoga a los servidores Linda. Estos publicadores se encargarán de publicar periódicamente tuplas "oferta" (generadas aleatoriamente) en el espacio de tuplas hasta que todos los buscadores hayan finalizado.

El gestor de buscadores comienza lanzando 10 threads buscadores simples y 5 threads buscadores combinados. Cada uno de ellos hace uso del Linda Driver para conectarse al registro de despliegue y posteriormente a los servidores Linda. Los buscadores se encargarán de leer ofertas durante un determinado número de iteraciones y mostrarlas por pantalla. Para ello, necesitan obtener primero una tupla "permiso" en cada iteración, devolviéndola al espacio una vez acabada.

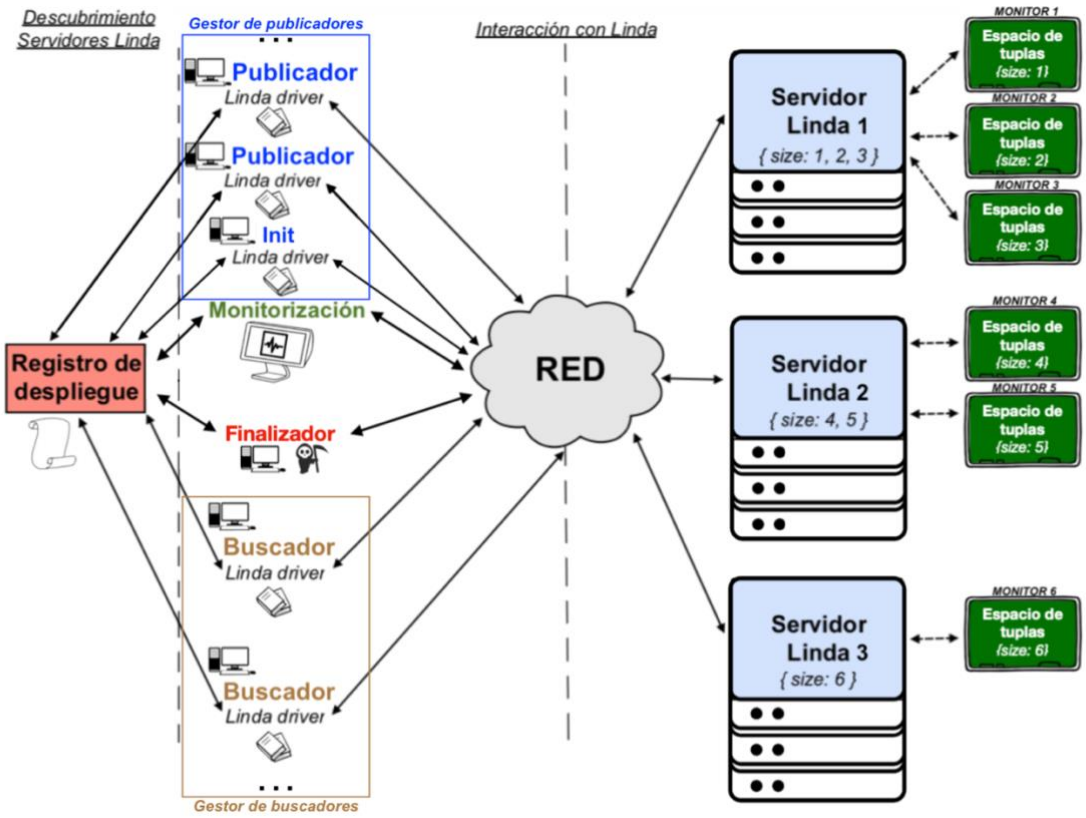


Figura 2: Esquema del sistema de gestión de ofertas de viaje

3. Decisiones de diseño relevantes

Nota. Los siguientes componentes del sistema, a excepción del registro de despliegue, del servidor de monitorización, y del cliente finalizador, requieren de una librería donde estén definidas las tuplas y sus operaciones.

3.1. Clientes

Se tratan de procesos que requieren de la dirección IP y el puerto del registro de despliegue como argumentos en su invocación para poder conectarse a él y así recibir información sobre la localización de los servidores Linda. El cliente hace uso de la librería Linda Driver para hacer llegar a estos servidores diversas peticiones de trabajo con tuplas (PN, RN, RDN, RN_2, RDN_2).

3.2. Linda Driver

Es la librería que usan los clientes para comunicarse con los servidores Linda. Está implementada como una clase. En el propio constructor, se conecta al registro de despliegue, obtiene las IP's y puertos de los servidores Linda (recibidos en un mensaje en formato IP:PORT:IP:PORT:IP:PORT, con cada par IP:PORT propio de un servidor Linda) y mantiene conectado al cliente a éstos. Después, responde al registro de despliegue con el mensaje "END_OF_SERVICE". El Linda Driver sabe de antemano que hay 3 servidores y también el tamaño de tupla que soporta cada uno. En el destructor, desconecta al cliente de todos los servidores y les envía el mensaje "END_OF_SERVICE". El Linda Driver ofrece las operaciones PN, RN, RDN, RN_2 y RDN_2, cuya función es enviar la respectiva petición al servidor Linda correspondiente. Para saber a qué servidor enviar la petición, el Linda Driver analiza el tamaño de las tuplas con las que desea trabajar el cliente.

3.3. Registro de despliegue

Es un servidor multithread (uno por cada conexión) invocado con los puertos de los servidores Linda. Su protocolo consiste primero, en leer de un fichero dado las direcciones IP de los servidores Linda. Tras la conexión de un cliente, envía a éste dicha información con el formato IP:PORT:IP:PORT:IP:PORT, con cada par IP:PORT propio de un servidor Linda. Después, espera a recibir un mensaje respuesta del cliente. Si el mensaje recibido es "END_OF_SERVICE", corta la comunicación con el cliente y espera a la conexión de otro cliente. Mientras que si el mensaje es "CLOSE_SERVER", corta la comunicación con el cliente y no acepta nuevas conexiones de clientes, acabando su ejecución.

3.4. Monitorización

Este servidor ofrece periódicamente por la salida estándar información estadística sobre el estado del sistema de coordinación. Se invoca con las IP's y el puerto del registro de despliegue como argumentos, para poder conectarse a él y obtener las IP's y puertos de los servidores Linda, al igual que los clientes. Tras esto, el servidor de monitorización lanza 3 thread y cada uno se conecta a un servidor Linda distinto. Estos thread se encargan de enviar a su respectivo servidor Linda una petición de información "DATA_REQUIRED" sobre todas las pizarras de tuplas con las que trabaja (número de tuplas, número de lecturas totales, número de tuplas borradas, número de lecturas pendientes y número de escrituras totales). Tras la respuesta del servidor, muestran esta información por pantalla, junto a la fecha y el tamaño de las tuplas de cada pizarra. Esta monitorización sigue el siguiente esquema:

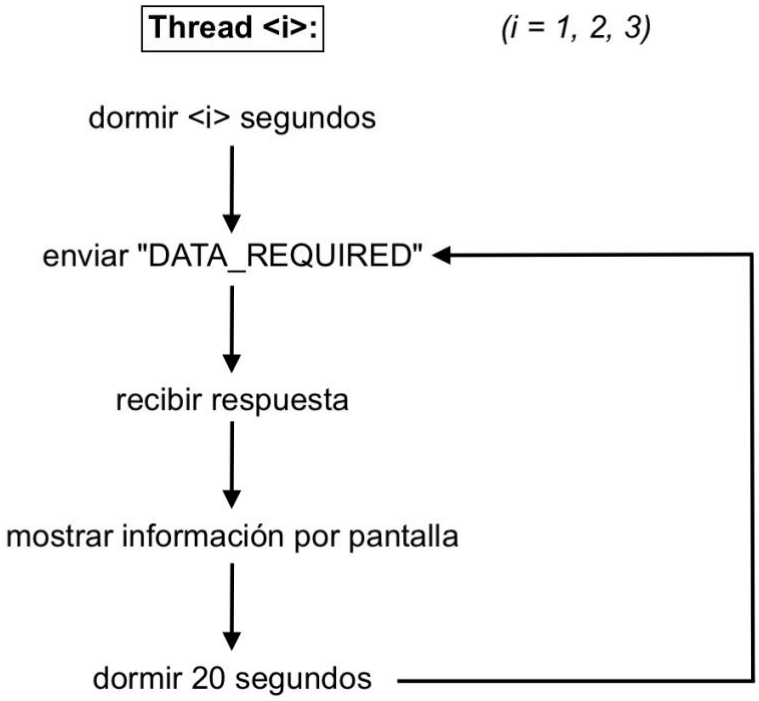


Figura 3: Esquema del comportamiento de cada thread conectado a un servidor Linda

Este servidor ha sido programado de forma que se puede ejecutar en una máquina independiente a las utilizadas por los tres servidores Linda, y acaba cuando se introduce el carácter 'q'.

3.5. Espacio de tuplas

Está representado por 6 pizarras independientes entre sí, una por cada tamaño de tupla. Cada pizarra está implementada como una lista dinámica doblemente enlazada de la biblioteca de C++. Es necesario el uso de un TAD no estático puesto que el número máximo de tuplas que puede albergar una pizarra no está determinado. Esta estructura de datos permite un cómodo almacenamiento de tuplas y acceso a las mismas, gracias a sus operaciones. También garantiza una optimización en el coste temporal de las operaciones que acceden al espacio. Dado que el sistema Linda debe ser no determinista, esta lista permite satisfacer esta propiedad gracias al fácil uso de su iterador.

3.6. Monitores Linda

Para gestionar la concurrencia en el acceso al espacio de tuplas, se construyen 6 monitores, uno por cada pizarra de tuplas. De esta forma, sólo un proceso puede acceder simultáneamente a una pizarra de tamaño determinado. Cada monitor contiene en la parte pública (además del constructor y destructor) las operaciones PN, RN, RDN, RN_2 y RDN_2 descritas en la introducción de este documento. Además, contienen una operación INFO para informar al servidor de monitorización sobre el estado del monitor, definido por sus variables permanentes principalmente. En cuanto a la parte privada, cada monitor cuenta con un mutex para la exclusión mutua; una variable condición <hayTupla>, para aquellos procesos que quieran leer una tupla del espacio, pero ninguna de las actuales hagan match con el patrón buscado; las variables permanentes <lectPend> (lecturas pendientes), <lectTotal> (lecturas totales), <borradas> (tuplas borradas) y <escTotal> (escrituras totales), que representan el estado la pizarra de tuplas; y finalmente, la lista de tuplas <espacioTup> descrita en el punto anterior.

A continuación, se describe el diseño de las operaciones de los monitores Linda. A excepción de INFO, todas ellas actualizan cuando corresponde las variables permanentes del monitor.

3.6.1. PN

Tiene como parámetro de entrada una tupla <t>. Añade <t> en una posición aleatoria de la lista y avisa a todos los procesos bloqueados en la cola de <hayTupla>.

3.6.2. RDN

Tiene una tupla <p> (patrón) como parámetro de entrada y una tupla <t> como parámetro de entrada/salida. Empieza a buscar de forma circular en una posición aleatoria de la lista. Si encuentra una tupla que haga match con <p>, la escribe en <t> y acaba. Si llega de nuevo a la posición inicial sin haber encontrado ninguna tupla que haga match, se bloquea en la cola de <hayTupla> y volverá a buscar cuando algún proceso le desbloquee (tras un PN).

3.6.3. RN

Mismo protocolo que RDN, pero borrando del espacio la tupla que haga match con <p>.

3.6.4. RDN_2

Tiene dos tuplas <p1> y <p2> como parámetros de entrada y dos tuplas <t1> y <t2> como parámetros de entrada/salida. Primero, localiza las posibles variables comunes de <p1> y <p2>, guardando sus posiciones.

```
string var := ""    // Para guardar la posible variable común
integer cont := 0   // Contador de variables comunes
integer array[1..t1.size()] := (1..t1.size(), 0) // Posiciones primeras variables comunes de p1
integer array[1..t2.size()] := (1..t2.size(), 0) // Posiciones todas las variables comunes de p2

for (integer i := 0; i < t1.size(); i := i + 1)  // Recorre p1
    var := p1.get(i) // var es el elemento i-ésimo de p1
    if (var es variable)  // Busca en p1 una variable
        for (integer j := 0; j < t2.size(); j := j + 1)  // Recorre p2
            if (var = p2.get(j) AND vars2[j] = 0)   // Busca en p2 todas las variables iguales a la
                                                    // encontrada en p1 que no hayan sido marcadas antes
                // Marca las posiciones de la variable común
                cont := cont + 1
                vars1[i] := cont
                vars2[j] := cont
            endif
        endfor
    endif
endfor
```

Figura 4: Diseño en pseudocódigo de la localización de variables comunes

Tras esto, busca todas las combinaciones posibles de pares de tuplas distintas hasta que se acaben las combinaciones o se encuentre el primer par que haga match con <p1> y <p2>. Para ello, primero busca match (como se ha descrito en RDN) para <p1>. Si no encuentra, se bloquea en la cola de <hayTupla>. Si encuentra, la escribe en <t1> y escribe en <t2> el patrón <p2>, pero ahora con el valor correspondiente de las variables comunes que contenga. Luego, busca match para <t2>. Si encuentra, la escribe en <t2> y acaba. Si no, vuelve a buscar otro match para <t1> y repite el procedimiento. Si se acaban las posibles combinaciones de pares de tuplas sin que ninguno haga match con <p1> y <p2> respectivamente, se bloquea en la cola de <hayTupla> hasta que otro proceso le desbloquee (tras un PN).

3.6.5. RN_2

Mismo protocolo que en RDN_2, pero borrando del espacio las tuplas que hagan match con <p1> y <p2> respectivamente una vez encontradas ambas.

3.6.6. INFO

Tiene un string <s> como argumento de entrada/salida. Modifica <s> concatenando información (en forma de string) sobre la pizarra de tuplas asociada al monitor. Esta información se compone por el número de tuplas en la pizarra y el valor de las variables permanentes del monitor.

3.7. Servidores Linda

Los servidores Linda son invocados con el puerto como argumento. Cada servidor construye los monitores necesarios para gestionar el acceso a sus pizarras y se prepara para atender clientes. Tras esto, lanza un thread por cada conexión a ellos. Cada thread atiende a un cliente, luego está constantemente esperando a recibir un mensaje. Una vez recibido, lo analiza. Si el mensaje es “END_OF_SERVICE”, el cliente será desconectado del servidor. Si el mensaje es "CLOSE_SERVER", el cliente será desconectado y el servidor no atenderá a más clientes. Si el mensaje es “DATA_REQUIRED”, se llamará a la operación de información de cada monitor del servidor. Si el mensaje es otro, entonces se trata de una petición de servicio. En este caso, el servidor llamará a la operación solicitada del monitor correspondiente en función del tamaño de tupla con el que se desee trabajar.

A modo de resumen, los threads lanzados por los servidores Linda siguen el siguiente esquema:

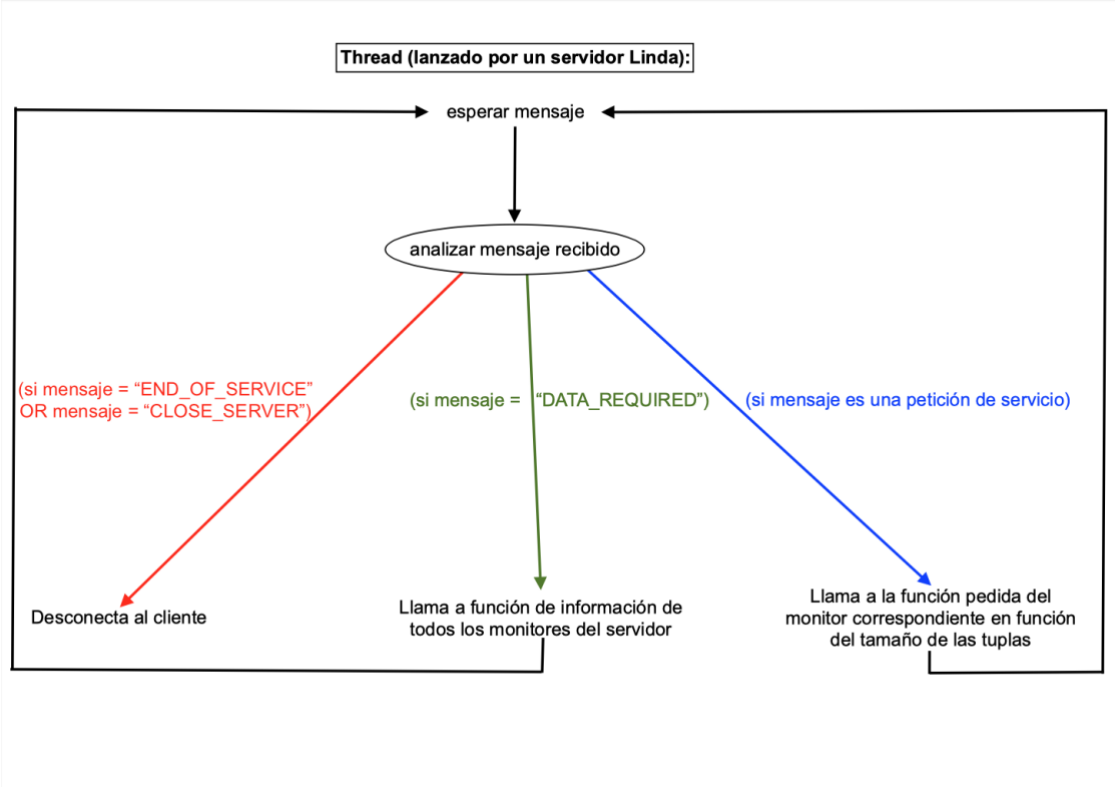


Figura 5: Esquema del comportamiento de cada thread lanzado por un servidor Linda

3.8. Cliente finalizador

Es un cliente especial que, al igual que los clientes normales, se conecta al registro de despliegue para obtener las IP’s y puertos de los servidores Linda. Sin embargo, en lugar de responder al registro con el mensaje “END_OF_SERVICE”, le responde con el mensaje de finalización “CLOSE_SERVER”. Una vez conectado a los servidores, les envía también el mensaje “CLOSE_SERVER”. Esto garantiza la finalización controlada del sistema.

3.9. Procesos para el problema de la gestión de viajes

Las ciudades disponibles son: Huesca, Zaragoza, Teruel, Logroño, Pamplona, Bilbao, Santander, Lleida, Barcelona y Madrid. Los precios disponibles oscilan entre los 20 $ y los 50 $.

3.9.1. Gestor de publicadores

Es invocado con la IP y puerto del registro de despliegue. Comienza comportándose como un publicador "init", es decir, se conecta a los servidores Linda mediante el Linda Driver y publica en el espacio de tuplas lo siguiente: 1 tupla "id" de valor 1, 8 tuplas "permiso" y 1 tupla "fin de servicio" de valor 0. Después, lanza 5 threads publicadores. Cada uno se conecta de la misma forma que el init a los servidores Linda. Mientras la tupla "fin de servicio" tenga valor menor que 15, es decir, que no hayan acabado su ejecución todos los procesos buscadores, cada publicador seguirá publicando ofertas. Estas publicaciones consisten en lo siguiente: el publicador toma la tupla "id" del espacio, genera una tupla "oferta" con ciudad origen, ciudad destino y precio aleatorio, la publica en el espacio y devuelve la tupla "id" incrementada en 1 al espacio. Entre cada publicación, el publicador espera 5 segundos.

En ciertas historias de ejecución, los buscadores combinados podrían no finalizar su ejecución. Para corregir esto, cada oferta con ID múltiplo de 20 será generada de forma que la ciudad origen coincida con la ciudad destino de la oferta anterior.

3.9.2. Gestor de buscadores

Es invocado con la IP y puerto del registro de despliegue. Comienza lanzando 10 threads buscadores simples y 5 threads buscadores combinados. Cada buscador se conecta mediante el Linda Driver al registro de despliegue y posteriormente a los servidores Linda. Cada buscador simple realiza 10 iteraciones de búsqueda de ofertas, mientras que los buscadores combinados realizan 5 iteraciones.

En cada iteración, un buscador simple buscará entre 1 y 3 ofertas de id's consecutivos (el primer id se busca de forma aleatoria). Para ello, primero intenta tomar una tupla "permiso" del espacio. Si no hay, se bloquea hasta que haya. Si la consigue, lee dichas ofertas y muestra su información por pantalla con el formato "Viaje: origen->destino, precio". Antes de terminar la iteración, devuelve la tupla "permiso" al espacio.

Por otro lado, un buscador combinado en cada iteración busca dos ofertas que le permitan viajar de una ciudad a otra (elegidas aleatoriamente) pasando por una intermedia. Si las encuentra, muestra su información por pantalla con el formato "Viaje: origen->intermedia, precio" e "intermedia->destino, precio" en líneas distintas. Al igual que un buscador simple, necesita obtener una tupla "permiso" al principio de cada iteración y devolverla al final de la misma.

Una vez terminadas todas las iteraciones, el buscador (simple o combinado) toma la tupla "fin de servicio", incrementa su valor en 1 y la devuelve al espacio.

4. Evaluación del sistema final

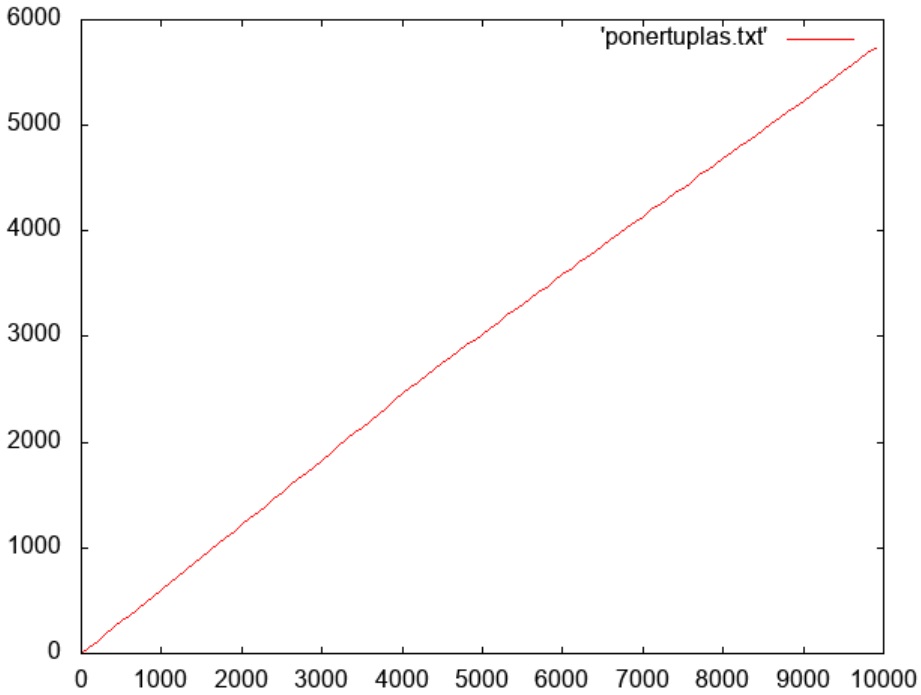
4.1. Despliegue del sistema

Para la evaluación final del sistema, sus elementos han sido desplegados en distintas máquinas.

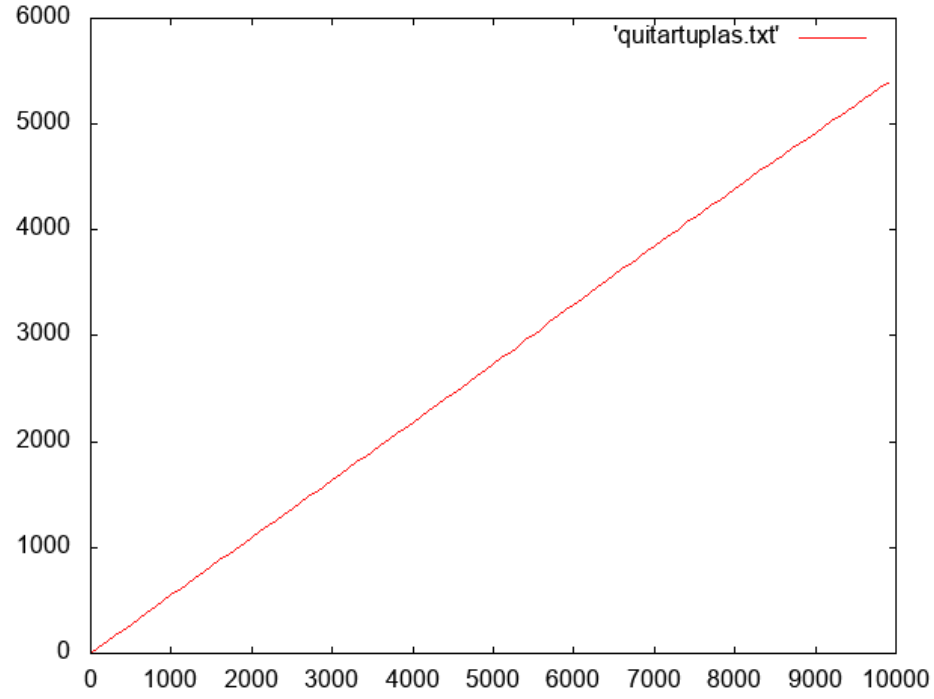
	RAM del ordenador	Procesador del ordenador	Sistema operativo del ordenador	Servidor o máquina virtual
Servidores Linda	8 GB	2,8 GHz 4 núcleos Intel Core i5	macOs BigSur 11.1	pilgor.unizar.cps.es
Registro de despliegue y servidor de monitorización	16 GB	1,19 GHz Intel Core i5-1035G1	Microsoft Windows 10 Home	hendrix.unizar.cps.es
Publicadores y buscadores	8 GB	2,1 GHz 4 núcleos AMD Ryzen 5 3550H	Microsoft Windows 10 Home	Ubuntu
Cliente carga y finalizador	16 GB	2,3 GHz Intel Core i7-10510U	Microsoft Windows 10 Home	WSL2

4.2. Rendimiento del sistema

Para comprobar el rendimiento del sistema, se ha realizado un experimento de carga con el ejecutable correspondiente a clienteCarga.cpp, un programa que inserta un gran número de tuplas en el espacio y luego las extrae. Dicho programa ha sido modificado para guardar en un fichero el tiempo que tarda en insertar o extraer las tuplas y generar las respectivas gráficas. En estas gráficas se puede apreciar cómo la implementación del sistema garantiza un coste lineal en el número de tuplas.



Gráfica 1: Coste temporal de la inserción de tuplas en función de su número



Gráfica 2: Coste temporal de la eliminación de tuplas en función de su número

5. Planificación y organización del trabajo

5.1. Decisiones organizativas y planificación temporal

La planificación y organización del trabajo son dos de las partes fundamentales del mismo. Se ha optado por llevar a cabo los pasos previos a la implementación correspondientes a la planificación de un diseño incremental.

En primer lugar, se ha realizado un listado de tareas a completar. Posteriormente, cada integrante del equipo de desarrollo ha estimado el tiempo de ocupación de cada tarea. Esto ha permitido realizar una media de los tiempos estimados y facilitar la distribución y enfoque de las mismas.

En segundo lugar, se han organizado las tareas de forma incremental y por parejas, de acuerdo a las estimaciones temporales calculadas.

PRIMER REPARTO

A fecha 22/12/2020 con punto de control el 28/12/2020.

1 - Funciones de tuplas (Héctor, Javier).

2 - Diseño de las funciones de los servidores Linda (Carlos, Pablo).

3 - Primeras ideas para la gestión de viajes (Héctor, Javier).

4 - Implementación del Linda Driver (Luis, Santiago).

5 - Diseño del registro de despliegue (Luis, Santiago).

SEGUNDO REPARTO

A fecha 28/12/2020 con punto de control el 02/01/2021.

1 - Implementación de las funciones de los servidores Linda y del monitor Linda (Carlos, Pablo).

2 - Implementación del Linda Driver (Luis, Santiago).

3 - Diseño e implementación de la Monitorización (Héctor, Javier).

4 - Implementación del registro de despliegue (Luis, Santiago).

REUNIÓN DE SINCRONIZACIÓN

Dedicada a la puesta en común de los avances y depuración de errores a fecha 02/01/2021.

TERCER REPARTO

A fecha 03/01/2021 con punto de control el 06/01/2021.

1 - Diseño de la solución a la gestión de viajes (Héctor, Javier).

2 - Integración de la monitorización y del registro de despliegue (Carlos, Santiago).

3 - Presentación del trabajo (Santiago).

4 - Memorias técnicas y de pruebas (Luis, Pablo).

ACTUALIZACIÓN TERCER REPARTO

A fecha 06/01/2021 con objetivo de tener el proyecto acabado y a pleno rendimiento el 13/01/2021.

1 - Implementación de los procesos publicadores y buscadores (Héctor, Javier).

2 - Integración de todos los componentes del sistema (Carlos).

3 - Presentación del trabajo (Santiago).

4 - Memorias técnicas y de pruebas (Luis, Pablo).

5 - Estrategia de finalización controlada (Carlos).

5.2. Esfuerzos dedicados por cada miembro del equipo

A grandes rasgos, estos son los esfuerzos realizados por cada integrante del equipo:

Javier Pardos Blesa: ilustraciones, tuplas, monitorización, cliente y ejercicio gestión de viajes.

Luis Palazón Simón: Linda Driver, registro de despliegue, documentación en memorias y material gráfico.

Héctor Toral Pallás: ilustraciones, tuplas, monitorización, cliente y ejercicio agencia de viajes.

Carlos Paesa Lía: servidores Linda, monitor Linda, integración de avances y estrategia de finalización.

Pablo Jesús Bueno Ereza: servidor Linda, monitor Linda, documentación en memorias y preparación de la defensa.

Santiago Illa Criado: Linda driver, registro de despliegue, integración de avances, presentación del trabajo y preparación de la defensa.

6. Conclusiones y posibles mejoras futuras

6.1. Decisiones que consideramos acertadas

Entre las decisiones que se consideran acertadas cabe destacar la planificación incremental, el trabajo por parejas durante la mayor parte del proyecto (cambiando a trabajo en tríos en la fase final del mismo), y el uso de hitos de control a nivel organizativo.

Además, el uso de una lista doblemente enlazada de la biblioteca de C++ como representación del espacio de tuplas y la elección de un monitor como herramienta de sincronización ha sido conveniente a nivel de programación.

6.2. Decisiones que consideramos mejorables

Una distribución de tareas más uniforme en el tiempo habría reducido bastante la carga de trabajo en las semanas previas a la entrega del proyecto. Además, haber comprendido perfectamente el enunciado el trabajo antes de las primeras reuniones habría agilizado el desarrollo de las tareas.

Como mejoras a añadir, el equipo valora que una generalización del Linda Driver, para poder comunicarse con servidores de cualquier tamaño de tuplas haría más completa la solución.

6.3. Conclusión

En conclusión, el equipo considera que el problema planteado ha sido resuelto en su totalidad, mostrando resultados convincentes y de muy bien rendimiento.